Final Technical Report to the

Office of Naval Research

by

Jeffery L. Kennington
Department of Computer Science and Engineering
SOUTHERN METHODIST UNIVERSITY
School of Engineering and Applied Science
Dallas, Texas 75275 (214) 768−3088
jlk@seas.smu.edu

for

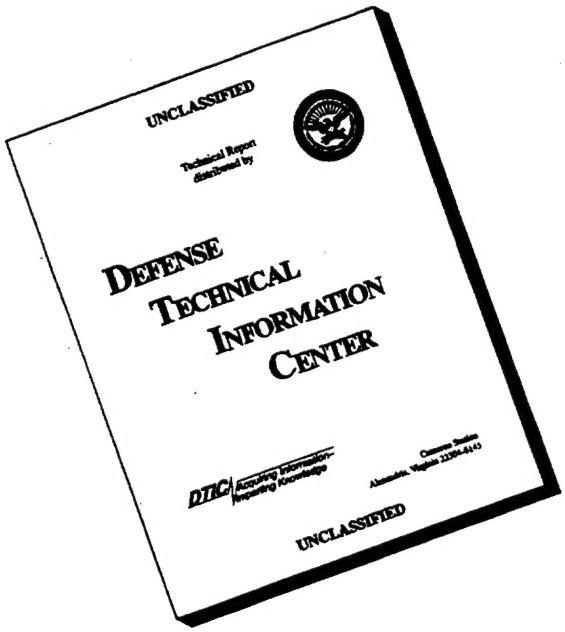
Real Time Optimization: Algorithms and Applications

ONR Contract Number N00014−95−1−0645

SMU Number 5−25175

30 August 1996

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	unrestricted
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Southern Methodist Univ. / 6b. OFFICE SYMBOL (If applicable): CSE	7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code): 6425 Airline Drive, Dallas TX 75275-0122	7b. ADDRESS (City, State, and ZIP Code): 800 North Quincy Street, Arlington VA 22217-5660
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: ONR / 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code): 800 North Quincy Street, Arlington VA 22217-5660	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

12. PERSONAL AUTHOR(S): Jeffery L. Kennington

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
technical	FROM 4/1/95 TO 8/31/96	8/30/96	110

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	assignment problem, generalized networks, class scheduling

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document contains three technical reports. The first report presents a new branch–and–bound algorithm for the assignment problem with side constraints. Models of this type are used in studies involving the optimal assignment of sailors to billets. Problems having 300 sailors and 600 billets with 90 potential assignments for each sailor were solved in less than five minutes on a Dec Alpha workstation. The second report presents several new algorithms for the problem of developing an annual class schedule for either a Navy C–School or a Navy A–School. Algorithms based on a greedy heuristic were found to perform very well on problems for both types of schools. The final report presents a new dual simplex based algorithm for the generalized network problem. In empirical tests, our specialized code was found to be approximately twenty times faster than CPLEX 3.0.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Jeffery L. Kennington	22b. TELEPHONE (Include Area Code): (214) 768-3088 / 22c. OFFICE SYMBOL: CSE

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
unclassified

Table of Contents

I. Statement of Work

During the 1970's most optimization models were run in batch mode on a large mainframe and were used to help solve some type of planning problem. During the 1980's we saw more demand for optimization models which were components of some real time system. That is, decisions were being made sequentially and some optimization model was communicating with an on−line database and was being used to provide information for real time decision making. In the 1990's, clients are demanding real time systems for problems which were formerly solved in batch mode.

In the area of assigning sailors to technical schools (both C and A Schools) and assigning sailors to job billets, the Navy is already developing on−line computer systems. The Job Advertising and Selection System (JASS) runs on−line and allows a career counselor on a ship to down load billet information from a server located in Washington D.C. A version of JASS which contains an optimization module is being developed and our research group developed the optimization software which is being incorporated into the system. We expect additional demand for high speed optimization software in the area of Navy personnel assignment and our work on the constrained assignment problem is related to this demand.

The Navy is currently developing a new reservation system to improve the assignment of its 400,000 sailors to C and A training schools. To help insure the success of the new reservation system, modern mathematical models and computational tools are needed to help generate the class schedules offered by the various training schools. The problems are members of the class NP−Hard and require special heuristic algorithms for real time application. Our work on efficient algorithms for class scheduling is related to this new reservation system.

Other Navy applications involve real time targeting of cruise missiles. Generally a missile is called a cruise missile if its speed is sub−sonic, if it uses a built−in global positioning navigation system (GPS/INS), and if its range is at least several hundred miles. A specific mission for a cruise missile is programmed by specifying a sequence of co−ordinates. The missile uses its on−board computational facilities and its GPS/INS system to guide it through this sequence of points. At present, once a missile is launched, its mission can not be modified. However, new versions are under consideration which can be redirected after launch. Finding new targets involves on−line optimization and our work on efficient primal and dual algorithms for generalized networks could support this effort.

II. Navy Personnel Assignment

After several years of down sizing, the Navy currently has approximately 400,000 sailors each of whom is assigned to a job billet for a tour of duty lasting from two to five years. Near the end of a tour (during the last six months) the sailor is placed into an assignment pool of sailors who are eligible for a new assignment. Sailors assigned to sea billets are usually rotated to shore billets and those stationed on the shore are usually assigned to a sea billet. Some 200 detailers within the Bureau of Navy Personnel are responsible for making the actual assignments. Every one that has studied this system agrees that Navy personnel assignment is a very complex process which involves trade–offs among conflicting policies. It is also complicated by the fact that there are many more billets than sailors.

Over the last decade, operations research analysts at the Navy Personnel Research and Development Center have developed several optimization models for various studies. Many of these optimization models are binary integer programming problems with special structure. The underlying structure which occurs most frequently is that of an assignment problem with a few linear side constraints. The side constraints generally involve a budget constraint on PCS (permanent change of station) cost and may involve quotas on seats at technical schools.

Our research team has developed a special algorithm for the constrained assignment problem. The algorithm is based on branch–and–bound and uses a Lagrangean relaxation to improve the lower bound. A complete description of this algorithm appears in Appendix A.

III. Class Scheduling For Navy Training Schools

Before assuming a new assignment, a sailor frequently is required to complete one or more Navy training classes. Experienced sailors (E4's through E9's) attend advanced skills training courses while new recruits (E1's, E2's, and E3's) take a set of basic courses. The advanced courses last from one week to over six months and are offered by Navy C Schools. The basic courses are generally shorter and are offered by Navy A Schools. Over 2000 different courses are taught each year at a cost of over $1.3 billion.

Navy School managers develop annual class offerings which are then used by the detailers in making billet assignments. Until recently, the class schedules were prepared manually by personnel at each School. Due to budget and instructor reductions, it is important that these Navy Schools operate as efficiently as possible making optimal use of their resources (facilities and instructors).

For C–Schools the complicating feature is that many sailors who attend the School need several different courses. A sailor only attends one course during a n week period, then he/she would go to another course. Ideally if a sailor needed four courses, then they would be scheduled back–to–back. This would reduce the idle time between courses.

The A–School courses are team taught with different numbers of instructors required on different days of a given course. The objective is to develop a feasible schedule that uses the smallest number of instructors on the busiest day of the year.

Algorithms for these scheduling problems have been developed and tested on real data. The results of our study may be found in Appendix B.

IV. Generalized Networks

There are many applications in the area of network flows where the flow either increases or decreases as it passes along an arc. In the area of Navy unit personnel readiness, promotions and attrition can be modelled using this feature.

We developed a new optimizer for this model that has both a primal and dual simplex implementation. In empirical tests, we found that the new optimizer is approximately twenty times faster than CPLEX 3.0 on moderate sized problems. A summary of our investigation may be found in Appendix C.

Appendix A

The Constrained Assignment Problem

Technical Report 93-CSE-49

A Truncated Exponential Algorithm for the Lightly Constrained Assignment Problem

by

Jeffery L. Kennington
and
Farin Mohammadi

Department of Computer Science and Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275-0122

Revised May 1996

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

This manuscript presents a truncated branch–and–bound algorithm to obtain a near optimal solution for the constrained assignment problem in which there are only a few side constraints. At each node of the branch–and–bound tree a lower bound is obtained by solving a singly constrained assignment problem. If needed, Lagrangean relaxation theory is applied in an attempt to improve this lower bound. A specialized branching rule is developed which exploits the requirement that every man be assigned to some job. A software implementation of the algorithm has been tested on problems with five side constraints and up to 75,000 binary variables. Solutions guaranteed to be within 10% of an optimum were obtained for these 75,000 variable problems in from two to twenty minutes of CPU time on a Dec Alpha workstation. The behavior of the algorithm for various problem characteristics is also studied. This includes the tightness of the side constraints, the stopping criteria, and the effect when the problems are unbalanced having more jobs than men.

ACKNOWLEDGMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR F49620–93–1–0091, the Office of Naval Research under Contract Numbers N00014–91–J–1234 and N00014–95–1–0645, and the Navy Personnel Research and Development Center.

I. INTRODUCTION

The constrained assignment problem is to determine a least cost assignment of m people to n jobs such that an additional set of constraints is satisfied. This model is a binary linear program and may be stated mathematically as follows:

$$\text{minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j:(i,j)\in A} x_{ij} = 1 \ , \ i = 1,\ldots,m \tag{2}$$

$$\sum_{i:(i,j)\in A} x_{ij} \leq 1 \ , \ j = 1,\ldots,n \tag{3}$$

$$x_{ij} \in \{0,1\}, \text{ all } (i,j) \in A \tag{4}$$

$$\sum_{(i,j)\in A} d_{ij}^{k}x_{ij} \leq r^{k}, \quad k = 1,\ldots,s \tag{5}$$

where $x_{ij} = 1$ implies that person i is assigned to job j at cost of c_{ij}, d_{ij}^{k} denotes the coefficient of x_{ij} in the kth side constraint, r^{k} denotes the right−hand−side for the kth side constraint, and A is the set corresponding to the feasible assignments. Note that the problem allows for more jobs than people. Many practical problems have this feature. It also allows for the case in which the number of people exceeds the number of jobs. For this case, one simply reverses the definition of people and jobs.

The problem (1)−(4) with m=n is the classic sparse assignment problem which is also known as the bipartite matching problem. This special model is a member of the

class P and there are excellent algorithms and software implementations of these algorithms available. We use one of these software implementations as a subroutine in this investigation.

Since (1)−(5) is a binary linear program, all the literature on integer programming applies (see Geoffrion and Marsten [1972], Salkin [1974], Parker and Rardin [1988], Nemhauser and Wolsey [1988]). A special case in which the side constraints have the generalized upper bound (GUB) structure has been studied by Ali, Kennington, and Liang [1993]. A relaxation/decomposition procedure that involves solving a series of pure assignment problems is used successfully. Ball, Derigs, Hilbrand, and Metz [1990] also present an algorithm for the matching problem with generalized upper bound side constraints. Another special case for s=1 and m=n has been studied by Gupta and Sharma [1981], Aggarwal [1985], Bryson [1991], Kennington and Mohammadi [1994, 1995]. The only specialized algorithm for (1)−(5) is the two phase procedure of Mazzola and Neebe [1986]. The first phase uses subgradient optimization to obtain an advanced start for the branch−and−bound method used in the second phase.

This research was motivated by our work with the Navy Personnel Research and Development Center located in San Diego. This general model often appears in various analytical studies which involve the assignment of sailors to billets. Most of these applications are for the unbalanced model in which there are more jobs (billets) than sailors. The Navy has approximately 400,000 sailors which are periodically reassigned to different billets. Sailors rotate between sea billets and shore billets with various restrictions on these assignments. One important restriction involves the cost of moving a sailor and his/her family from one location to another. This is referred to as the cost for a permanent change of station and the corresponding side constraint is known as the

PCS budget constraint. Fleet balance between the Atlantic and Pacific Fleets is also important and leads to other side restrictions. The objective of this study was to develop a truncated branch–and–bound algorithm to solve the constrained assignment problem and to provide an empirical analysis of this algorithm on a variety of assignment problems having only a few side constraints.

II. THE GENERIC ALGORITHM

Consider the problem $P(S) \equiv \min\{ cx: x \in S\}$. Using the terminology of Geoffrion and Marsten [1972], $v[P(S)]$ denotes the optimal objective function value for $P(S)$, $\overline{P}(S)$ denotes a relaxation of $P(S)$, and CL denotes the candidate list. The generic branch−and−bound algorithm used in this investigation may be stated as follows:

Input:

1. The problem, $P(S)$.

Output:

1. The solution vector, x^*.
2. The objective value corresponding to x^*, v^*.($v^* = \infty$ implies that $S = \Phi$.)

Procedure BAB;

Begin

initialize:

CL:= {P(S)}, $v^* := \infty$;

while CL $\neq \Phi$ **do**

comment: select a candidate problem for analysis.

select $P(U) \in$ CL, CL:= CL \ {P(U)};

if $\overline{P}(U)$ **has a feasible solution, then**

if $v[\overline{P}(U)] < v^*$, **then**

let $\overline{x}$ **be an optimum for** $\overline{P}(U)$;

if $\overline{x} \in S$, **then**

$x^* := \overline{x}$, $v^* := c\overline{x}$;

else

apply a heuristic to $\bar{x}$ **in an attempt to create** $\hat{x}$ **such that** $\hat{x} \in S$ **and** $c\hat{x} < v^*$;

if successful, then $x^* := \hat{x}, v^* := c\hat{x}$;

comment: branching

create $U_1, U_2, ..., U_p$ **such that** $U_1 \cup U_2 \cup ... \cup U_p = U$ **and** $U_i \cap U_j = \Phi$ **for all** $i \neq j \in \{1, 2, ..., p\}$;

$CL := CL \cup \{P(U_1), P(U_2), ..., P(U_p)\}$;

end if

end if

end if

end while

end.

III. THE SPECIALIZED ALGORITHM

The specialized techniques developed for the model (1) − (5) are presented in this section. This includes a relaxation, a branching rule, and fathoming rules based upon the underlying assignment structure. These are combined to form a new truncated exponential algorithm for the constrained assignment problem.

3.1 The Relaxation

Let D denote the matrix corresponding to the coefficients in (5), x denote the vector corresponding to the binary decision variables, c denote the vector of costs, and r denote the vector of right−hand−side values for the side constraints. Then the constrained assignment problem may be denoted as P(S) where S = {x: (2), (3), (4), (5)}. Let $\underline{1}$ denote a vector of 1's and $\overline{S}$ = {x: (2), (3), (4), $\underline{1}Dx \leq \underline{1}r$}. Then P($\overline{S}$) is a valid relaxation for P(S). Application of the algorithm in Kennington and Mohammadi [1994a] to solve the singly constrained assignment problem will yield a lower bound for P(S), the optimal Lagrangean multiplier corresponding to the constraint $\underline{1}Dx \leq \underline{1}r$, and $\bar{x} \in \overline{S}$. If $\bar{x} \in S$, then $c\bar{x}$ is an upper bound for P(S).

Let $\hat{S}$ = {x: (2), (3), (4)}. Recall that a Lagrangean dual for P(S) is the problem max {$L(\alpha) : \alpha \geq 0$} where $L(\alpha)$ = min {$cx + \alpha(Dx - r): x \in \hat{S}$ }. We use the optimal Lagrangean multiplier and $\bar{x}$ from the singly constrained algorithm to form an advanced starting value for α. Let y denote the solution for $L(\alpha)$. Then $w = Dy - r$ is used to modify α for successive steps. A limited number of these steps will be performed in this algorithm.

3.2 The Branching Rule

Consider any node in the branch−and−bound tree. If the relaxation $P(\bar{S})$ has no feasible solution, then this node may be fathomed. Otherwise, an assignment will be used to create the branches as illustrated in Figure 1.

Figure 1 here

For a given node in the branch−and−bound tree let $F^1 = \{(i,j): \bar{x}_{ij} = 1\}$ and $F^0 = \{(i,j): \bar{x}_{ij} = 0\}$. Let $\overline{U} = \{x \in \bar{S}: x_{ij} = 1 \text{ for all } (i,j) \in F^1 \text{ and } x_{ij} = 0 \text{ for all } (i,j) \in F^0\}$. The relaxation solved at each node in the branch−and−bound tree is $P(\overline{U})$ which is a singly constrained assignment problem with some assignments fixed. Let $\bar{x} \in \overline{U}$, $T = \{(i,j): \bar{x}_{ij} = 1, (i,j) \notin F^1\}$, and $t = |T|$. Consider the $t+1$ subsets of $\overline{U}$ $(\overline{U}_1, \overline{U}_2, ..., \overline{U}_{t+1})$ created in the following manner:

$\overline{U}_1 = \{ x \in \overline{U}: x_{i_1 j_1} = 0, (i_1,j_1) \in T\}, T_1 = T \backslash \{(i_1,j_1)\}$;

$\overline{U}_2 = \{ x \in \overline{U}: x_{i_1 j_1} = 1, x_{i_2 j_2} = 0, (i_2,j_2) \in T_1\}, T_2 = T_1 \backslash \{(i_2,j_2)\}$;

$\overline{U}_3 = \{ x \in \overline{U}: x_{i_1 j_1} = 1, x_{i_2 j_2} = 1, x_{i_3 j_3} = 0, (i_3,j_3) \in T_2\}, T_3 = T_2 \backslash \{(i_3,j_3)\}$;

.

.

.

$\overline{U}_t = \{ x \in \overline{U}: x_{i_1 j_1} = 1, ..., x_{i_{t-1} j_{t-1}} = 1, x_{i_t j_t} = 0, (i_t, j_t) \in T_{t-1}\}$;

$\overline{U}_{t+1} = \{ x \in \overline{U}: x_{ij} = 1 \text{ for all } (i,j) \in T\}$. Note that, $\overline{U} = \overline{U}_1 \cup \overline{U}_2 \cup ... \cup \overline{U}_{t+1}$ and $\overline{U}_i \cap \overline{U}_j = \Phi$ for all $i \neq j$.

Consider a node in the branch−and−bound tree having f_1 edges fixed at 1. Then branching from this node will produce $t+1 = n - f_1 + 1$ new candidate problems. From Figure 1 it can be seen that the last node (i.e., U_{t+1}) need not be created since it was examined at the parent node. Therefore, each branching produces t candidate problems.

3.3 The Candidate List

For our implementation of the algorithm, a singly constrained assignment problem will be applied to $P(\overline{U}_i)$ and the results placed in the candidate list (i.e., a problem is solved before it is placed in the candidate list). The motivation for placing solved problem in the candidate list is that the solution for $P(\overline{U}_i)$ can be easily modified to obtain an advanced starting solution for $P(\overline{U}_{i+1})$. Therefore solving the sequence of problems $P(\overline{U}_1), P(\overline{U}_2), ..., P(\overline{U}_t)$ should require only a moderate amount of computational effort. Hence, each entry in the CL consists of the five tuple $(F^0, F^1, \overline{x}, \overline{\beta}, u)$ where $\overline{x} \in \overline{U}_i$, $\overline{\beta} \leq v[P(\overline{U}_i)]$, and u the optimal Lagrangean dual for the singly constrained problem.

3.4 Fathoming Rules

At any node p of the branch−and−bound tree let U, F^1, and F^0 be as defined in Section 3.2. Let $M = \{1, 2, 3,..., m\} \setminus \{i: (i,j) \in F^1\}$, then the following rules may be used to fathom a node. If

$$\sum_{(i,j)\in F^1} d_{ij}^k + \sum_{i\in M} \min(d_{ij}^k \ : \ (i,j) \in A \setminus F^0) > r^k$$

for any k, then node p can be fathomed. That is, no selection of the free variables will satisfy the kth side constraint. If

$$\sum_{(i,j)\in F^1} c_{ij} + \sum_{i\in M} \min(c_{ij} \ : \ (i,j) \in A \setminus F^0) > v^*$$

then node p can be fathomed. That is, no selection of the free variables will result in a solution superior to the incumbent.

If $\min \{\underline{1}Dx: x \varepsilon U\} > \underline{1}r$, then node p can be fathomed. That is, no selection of the free variables will satisfy all side constraints simultaneously. Let β be the best lower

bound obtained for node p and ε be the termination tolerance. We will fathom node p if $v^* - \beta \leq \varepsilon\beta$. Using this rule with $\varepsilon = 0.1$ results in a solution from the procedure guaranteed to be within 10% of the optimum and $\varepsilon = 0.01$ produces a solution within 1% of an optimum. This rule speeds convergence at the expense of an exact solution.

3.5 The Algorithm

In this section the information presented in Sections 3.1–3.4 is used to construct a truncated exponential algorithm.

Input:

1. The cost vector, c.
2. The feasible region S.
3. The set of (man, job) pairs corresponding to eligible assignments, A.
4. Termination tolerance, ε.
5. The maximum execution time, tmax.
6. The maximum number of Lagrangean relaxations to be solved at each node, limit.

Output:

1. The solution vector, x^*.
2. The objective value corresponding to x^*, v^*. ($v^* = \infty$ implies that the problem is infeasible.)

Procedure ASSIGN+s;

Begin

initialize:

comment: Node 1 in the Branch–and–Bound tree.

$v^* := \infty$, $F^0 := \Phi$, $F^1 := \Phi$;

ASSIGNP1($P(\overline{S})$, $\bar{x}$, β, u);

if $\beta = -\infty$ **then terminate;**

if $\bar{x} \in S$ **then** $x^* := \bar{x}$, $v^* := c\bar{x}$;

else LAGRANGE($\bar{x}$, β, u);

comment: Tolerance test for fathoming.

if $v^* - \beta \leq \varepsilon\beta$ **then terminate;**

CL:= $\{(F^0, F^1, \bar{x}, \beta, u)\}$;

while CL $\neq \Phi$ **do**

SELECT A PROBLEM(F^0,F^1, $\bar{x}$, β, u);

BRANCH(CL, F^0,F^1, $\bar{x}$, β, u);

end while

end.

procedure ASSIGNP1($P(\overline{U})$, $\bar{x}$, β, u);

Begin

initialize:

$\beta := -\infty$;

apply the ASSIGN+1 algorithm (Kennington and Mohammadi [1994]) to $P(\overline{U})$;

if $P(\overline{U})$ **has a feasible solution then**

let $\bar{x}$ **be the best feasible solution found for** $P(\overline{U})$;

let β **be the best lower bound found for** $P(\overline{U})$;

let u **be the optimal Lagrangean dual for the singly constrained problem;**

end if

end.

procedure SELECT A PROBLEM $(F^0, F^1, \bar{x}, \beta, u)$;

Begin

select $(F^0, F^1, \bar{x}, \beta, u) \in CL$, $CL := CL \setminus (F^0, F^1, \bar{x}, \beta, u)$;

end.

procedure BRANCH$(CL, F^0, F^1, \bar{x}, \beta, u)$;

Begin

initialize:

$t := 0$, $G := \{(i,j): \bar{x}_{ij} = 1, (i,j) \notin F^1\}$;

$M := \{1, 2, 3, ..., m\} \setminus \{i: (i,j) \in F^1\}$;

while $G \neq \Phi$ **do**

comment: Fix a variable at zero.

let $(i_1,j_1) \in G$, $G := G \setminus \{(i_1,j_1)\}$, $F^0 := F^0 \cup \{(i_1,j_1)\}$;

ASSIGNP1$(P(\overline{U}), \bar{x}, \beta, u)$;

if $\beta \neq -\infty$ **then**

if $\bar{x} \in S$ and $c\bar{x} < v^*$ **then** $x^* := \bar{x}$, $v^* := c\bar{x}$;

else LAGRANGE$(\bar{x}, \beta, u)$;

comment: Tolerance test for fathoming.

if $v^* - \beta > \varepsilon\beta$ **then** $CL := CL \cup \{(F^0, F^1, \bar{x}, \beta, u)\}$;

comment: Permanently assign man i_1 to job j_1.

$M := M \setminus \{i_1\}$, $F^1 := F^1 \cup \{(i_1,j_1)\}$;

$F^0 := F^0 \cup \{(i_1,j) : (i_1,j) \in A \text{ for all } j\} \cup \{(i,j_1) : (i,j_1) \in A \text{ for all } i\} \setminus \{(i_1,j_1)\}$;

comment: Assignment polytope feasibility tests.

if $\sum_{(i,j)\in F^1} c_{ij} + \sum_{i\in M} \min(c_{ij} : (i,j) \in A) > v^*$ **then return;**

for k=1,...,s

if $\sum_{(i,j)\in F^1} d_{ij}^k + \sum_{i\in M} \min(d_{ij}^k : (i,j) \in A) > r^k$ **then return;**

end for

end if

end while

end.

Procedure LAGRANGE($\bar{x}$, β , u);

Begin

initialize:

$\alpha := u\underline{1}$, y:= $\bar{x}$, t:= 1;

while ($v^* - \beta > \varepsilon\beta$ **and** $t \leq$ limit) **do**

w:= Dy−r;

for i=1,...,s

if $w_k > 0$, **then** $\alpha_k := 1.25\alpha_k$;

if $w_k < 0$, **then** $\alpha_k := 0.75\alpha_k$;

end for

let y be a solution for $L(\alpha)$ **and** $\beta := \max\{\beta, v[L(\alpha)]\}$;

if $y \in S$ and $cy < v^*$ **then** $x^* := y$, $v^* := cy$;

t:= t+1;

end while

end.

This algorithm exploits the structure of the model (1)−(5) in several ways. Permanent assignment of a man to a job implies that all other variables involving this man and job may be fixed at zero. The relaxation is a singly constrained assignment problem for which near optimal integer solutions can be obtained using the results in Kennington and Mohammadi [1994]. Special fathoming rules which are based upon the assignment polytope are used.

IV. EMPIRICAL ANALYSIS

The specialized algorithm has been implemented in software (called ASSIGN+s) and empirically analyzed on an Alpha workstation by Digital Equipment Corporation. The code is written in Fortran and uses ASSIGN+1 (see Kennington and Mohammadi [1994]) to solve the singly constrained assignment problems. ASSIGN+1 is an implementation of the Lagrangean relaxation algorithm for sparse singly constrained assignment problems.

A test problem generator was developed which has the following inputs: (i) the number of men, (ii) number of jobs for each man, (iii) the maximum cost, $\bar{c}$, (iv) the number of side constraints, s, and (v) the side constraint multiplier, z. Both the costs and the side constraint coefficients are uniformly distributed over the range (0, $\bar{c}$). We randomly generate a feasible assignment, $\bar{x}$, and set the right−hand−side of the side constraints, r, to $zD\bar{x}$. Obviously, as z becomes smaller, the feasible region becomes smaller and for sufficiently small z {x: (2), (3), (4), and (5)} is usually empty.

The generator was used to generate two sets of 400x400 problems described in Table 1. As Table 1 indicates, the problems generally become more difficult as z becomes smaller and for z small enough the feasible region is empty. For all runs, the stopping criteria used is ε=10% and the % deviation reported in column 8 gives a guarantee on the deviation from optimality. All times are CPU time and exclude the time for both input and output. The run with problem number 2 having z=0.4 was terminated after the candidate list grew to 25,000 entries. As we expected, there exists problems which cannot be solved in a reasonable amount of time and storage using this approach. Tight-

ly constrained problems having 48,000 binary variables definitely stretches the capability of this software implementation.

Table 1 here

Tables 2 and 3 give our empirical results with 30 randomly generated assignment problems with various sizes all having five side constraints. For all of the problems tested we were able to find a solution guaranteed to be within 10% of an optimal solution. The six smallest problems have 3,000 binary variables. Five of these were solved in less than two minutes each and one required about six and one–half minutes. The six largest problems had 75,000 binary variables and were all solved in less than twenty one minutes each. The most difficult problems (300x300) have 27,000 binary variables. Two of these six problems required eighty minutes to solve. These two difficult problems also had very tight side constraints.

Tables 2 and 3 here

This work was motivated by models for assigning sailors to ships and for this application the number of jobs always exceeds the number of sailors available. Frequently the job list covers a longer period than the list of available sailors which produces a large imbalance in n and m. Tables 4 and 5 present our empirical results from solving 18 unbalanced assignment problems with five side constraints. For the 300x600 problem with $z=0.6$ presented in Table 5, the run was terminated due to candidate size limit. For all other test problems we were able to obtain a solution within 10% of an optimum.

Tables 4 and 5 here

For all test problems we search for a solution within 10% of an optima. To study the effect of the tolerance value on the performance of the algorithm we solved two 200x200 and two 200x400 problems with different tolerance values. Figure 2 indicates

that, as expected, a decrease in the tolerance value leads to an increase in the execution time. For all four problems, a point was reached in which a slight decrease in the tolerance resulted in a large increase in the solution time.

Figure 2 here

V. SUMMARY AND CONCLUSIONS

We have presented a truncated exponential algorithm for the constrained assignment problem. The algorithm is applicable for both balanced and unbalanced assignment problems having inequality side constraints. The algorithm uses a specialized branching rule that exploits the underlying structure of the problem. Bounds are obtained by solving a singly constrained assignment problem followed by a few iterations with a Lagrangean relaxation.

We present empirical results for both balanced and unbalanced problems having five side constraints. For problems having 75,000 binary variables, solutions guaranteed to be within 10% of an optima were obtained in less than twenty one minutes on a Dec Alpha workstation. Our analysis indicated that as the side constraints become tighter the execution time and number of branch–and–bound nodes increases. For one of the 300x300 problems having 27,000 arcs, the execution time increased from about one minute to about eighty minutes as a result of side constraint tightening. Our analysis also indicates that the performance of the algorithm on unbalanced problems is generally better than its performance for the balanced problems with the same number of binary variables. The Navy personnel assignment problems which motivated this study are all unbalanced models.

For problems of this type, having only a few side constraints, we believe that this is the current best algorithm and software implementation available. Solutions guaranteed to be within 10% of an optimum should be obtained for most problems having fewer than five side constraints and fewer than 20,000 arcs.

VI. REFERENCES

V. Aggarwal [1985], "A Lagrangean–Relaxation Method for the Constrained Assignment Problem," *Computers and Operations Research* vol. 12 pp. 97–106.

I. Ali, J. Kennington, and T. Liang [1993], "Assignment with En Route Training of Navy Personnel," *Naval Research Logistics Quarterly* vol. 40 pp. 581–592.

M. Ball, U. Derigs, C. Hilbrand, and A. Metz [1990], "Matching Problems with Generalized Upper Bound Side Constraints," *Networks* vol. 20 pp. 703–721.

N. Bryson, [1991] "Parametric Programming and Lagrangian Relaxation: The Case of the Network Problem with a Single Side–Constraint," *Computers and Operations Research* vol. 18 pp. 129–140.

A. Geoffrion and R. Marsten [1972], "Integer Programming Algorithms: A Framework and State–Of–The–Art Survey," *Management Science* vol. 18 pp. 465–491.

A. Gupta and J. Sharma [1981], "Tree Search Method for Optimal Core Management of Pressurized Water Reactors," *Computers and Operations Research* vol. 8 pp. 263–269.

J. Kennington and F. Mohammadi [1994], "The Singly Constrained Assignment Problem: A Lagrangean Relaxation Approach," *Computational Optimization and Applications*, vol. 3 pp. 7–26.

J. Kennington and F. Mohammadi [1995], "The Singly Constrained Assignment Problem: An AP Basis Approach," *Computational Optimization and Applications*, vol. 4 pp. 347–374.

J. Mazzola and A. Neebe [1986], "Resource Constrained Assignment Scheduling," *Operations Research* vol. 34 pp. 560–572.

G. Nemhauser and L. Wolsey [1988], *Integer and Combinatorial Optimization*, John Wiley and Sons: New York, NY.

R. Parker and R. Rardin [1988], *Discrete Optimization*, Academic Press Incorporated: New York, NY.

H. Salkin [1974], *Integer Programming*, Addison–Wesley Publishing Company: Reading, Massachusetts.

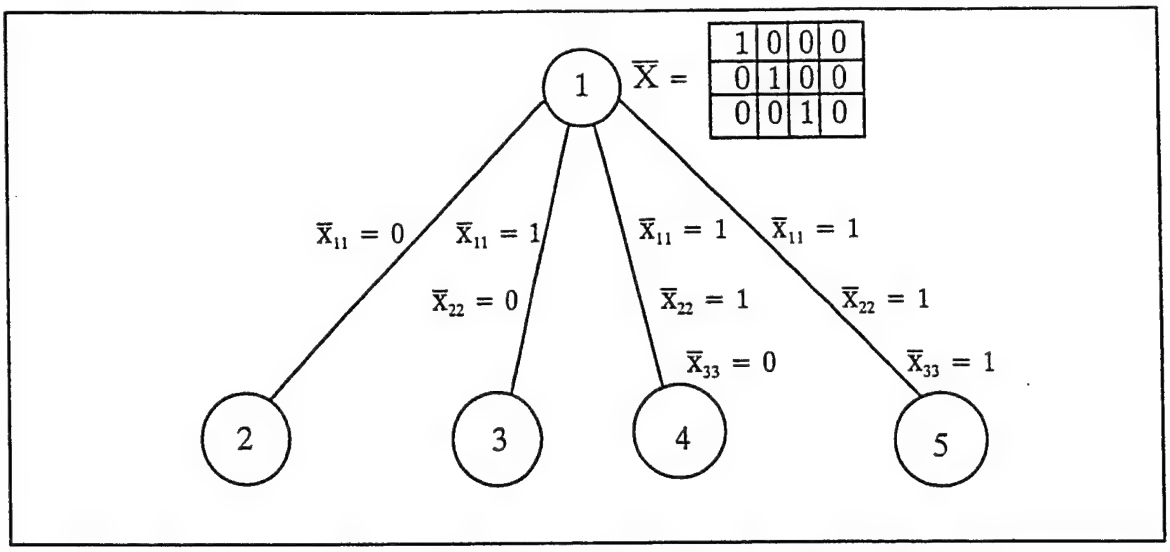
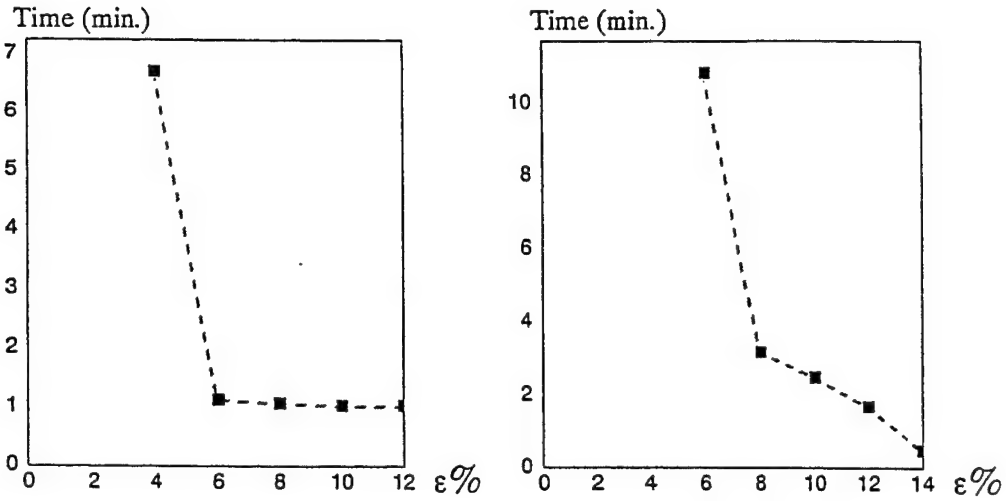
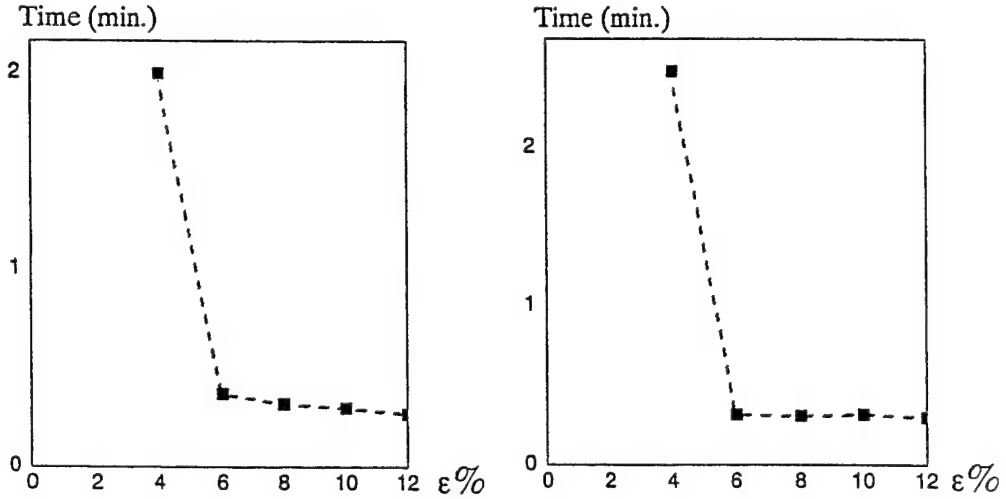


Figure 1. Example of branching rule for a 3x4 problem.



a. Problem #1 (200x200) with 1200 arcs

b. Problem #2 (200x200) with 1200 arcs



c. Problem #3 (200x400) with 1200 arcs

d. Problem #4 (200x400) with 1200 arcs

Figure 2. Plots of time versus the stopping tolerance for four problems.

Table 1. Empirical results from the algorithm for 400x400 assignment problems with five side constraints (48,000 columns and 805 rows).

Prob.	z	# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	Per Cent Deviation	Node # of Incumbent
1	1.0	189	216	0.71	5,270	5,309	0.74	13
	0.9	244	1,958	4.74	5,505	5,768	4.78	22
	0.8	274	2,119	3.96	6,999	7,271	3.89	78
	0.7	952	8,288	12.66	10,991	11,725	6.68	859
	0.6	14,142	124,222	166.00	20,820	22,713	9.09	13,943
	0.5	4,214	35,362	45.39	47,446	50,343	6.11	4,157
	0.4	1	2	0.00	problem has no feasible solution			
2	1.0	224	503	1.66	5,370	5,559	3.52	2
	0.9	253	1,801	4.57	5,830	6,132	5.23	34
	0.8	700	5,434	9.26	7,815	8,373	7.13	492
	0.7	3,114	25,786	42.93	12,510	13,649	9.10	2,938
	0.6	3,094	28,694	40.92	23,606	25,188	6.70	2,871
	0.5	3,505	29,642	31.45	50,939	54,690	7.36	3,340
	0.4	25,000[1]	79,160	53.37	148,825	no feasible solution obtained		
	0.3	1	2	0.0	problem has no feasible solution			

[1] terminated due to candidate list size limit.

Table 2. Empirical results with problem set 1 using the algorithm.
(All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	Per Cent Deviation	Node # of Incumbent
n x m	Jobs/Man	z							
100x100	30	1.0	215	859	0.06	5,021	5,356	6.67	178
	30	0.8	447	3,730	0.28	6,758	7,090	4.90	404
	30	0.6	11,938	100,799	6.60	18,726	20,599	10.00	5,686
200x200	60	1.0	111	192	0.13	5,131	5,153	0.43	4
	60	0.8	707	5,735	2.04	6,890	7,485	8.62	572
	60	0.6	5,832	52,193	15.78	22,314	24,336	9.95	5779
300x300	90	1.0	184	526	0.82	5,533	5,641	1.95	12
	90	0.8	240	1,892	1.87	7,559	8,006	5.90	78
	90	0.6	12,679	115,382	79.38	21,901	24,062	9.87	12,538
400x400	120	1.0	218	509	1.72	5,160	5,308	2.87	12
	120	0.8	325	2,541	4.63	7,363	7,954	8.03	146
	120	0.6	503	4,782	9.09	21,567	23,614	9.49	143
500x500	150	1.0	265	569	3.37	5,170	5,405	4.55	5
	150	0.8	397	3,421	10.70	7,481	7,833	4.70	139
	150	0.6	610	5,873	20.61	23,385	24,780	5.96	241

Table 3. Empirical results with problem set 2 using the algorithm.
(All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	Per Cent Deviation	Node # of Incumbent
n x m	Jobs/Man	z							
100x100	30	1.0	76	413	0.04	4,975	5,383	8.18	4
	30	0.8	1.394	12,757	1.002	7,974	8,598	7.82	1,337
	30	0.6	2,932	25,891	1.86	21,253	22,756	7.07	2,871
200x200	60	1.0	103	196	0.13	5,873	5,928	0.85	6
	60	0.8	1	8	0.00	8,010	8,783	9.64	1
	60	0.6	1,457	13,308	4.03	24,194	25,954	7.27	1,374
300x300	90	1.0	158	290	0.56	5,237	5,2800	0.82	10
	90	0.8	490	4,115	4.34	7,658	7,894	3.08	282
	90	0.6	10,813	99,948	79.77	23,985	26,055	8.63	10,659
400x400	120	1.0	250	1,126	2.78	4,892	5,160	5.46	18
	120	0.8	360	3,071	5.26	7,923	88,606	8.61	17
	120	0.6	2,476	23,947	39.17	25,8821	28,297	9.59	2,362
500x500	150	1.0	336	1,731	7.50	4,933	5,004	1.42	138
	150	0.8	410	3,385	9.69	7,968	8,514	6.85	134
	150	0.6	630	6,254	19.72	25,838	27,683	7.14	228

Table 4. Empirical results with problem set 3 using the algorithm.
(All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	Per Cent Deviation	Node # of Incumbent
n x m	Jobs/Man	z							
100x200	30	1.0	229	1,185	0.07	3,668	3,940	7.39	183
	30	0.8	90	883	0.04	5,384	5,800	7.70	76
	30	0.6	2,726	24,147	0.85	15,302	16,430	7.37	2,724
200x400	60	1.0	44	173	0.06	3,491	3,559	1.95	14
	60	0.8	144	1,556	0.28	5,038	5,224	3.68	142
	60	0.6	248	2,318	0.34	16,990	18,519	9.00	241
300x600	90	1.0	105	417	0.30	3,800	3,937	3.61	2
	90	0.8	271	2,262	0.87	5,297	5,674	7.11	62
	90	0.6	1,200	12,151	4.10	15,809	17,063	7.93	1,174

Table 5. Empirical results with problem set 4 using the algorithm.
(All problems have five side constraints.)

Problem Description			# BAB Nodes	# AP's Solved	Time (min.)	LB	UB	Per Cent Deviation	Node # of Incumbent
n x m	Jobs/Man	z							
100x200	30	1.0	47	238	0.02	3,333	3,445	3.36	5
	30	0.8	127	1,247	0.06	5,031	5,490	9.11	74
	30	0.6	9,651	93,433	3.74	14,160	14,650	3.46	9,589
200x400	60	1.0	1	5	0.00	4,332	4,629	6.86	1
	60	0.8	139	1,459	0.28	6,100	6,475	6.14	57
	60	0.6	183	1,933	0.30	18,080	19,303	6.76	131
300x600	90	1.0	101	401	0.24	3,644	3,729	2.50	37
	90	0.8	223	2,177	0.81	5,403	5,551	2.74	162
	90	0.6	25,000	256,241	89.84	17,537	20,611	17,53[1]	15,015

[1] terminated due to candidate size limit

Appendix B

Class Scheduling for Navy Training Schools

Technical Report 96–CSE–6

Class Scheduling Algorithms for Navy Training Schools

by

A. Apte[1]
A. Jayasuriya[1]
J. Kennington[1]
I. Krass[2]
R. Mohamed[3]
S. Sorensen[4]
and
J. Whitler[1]

[1]CSE Department, SMU, Dallas, TX 75275–0122

[2]Defense Manpower Data Center, Monterey, CA 93490–2453

[3]SABRE Decision Technologies, Dallas/Fort Worth Airport, TX 75261–9616

[4]Navy Personnel Research & Development Center, San Diego, CA 92152–6800

Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275–0122

June 1996

Abstract

The problem of developing good schedules for Navy C–Schools has been modeled as a combinatorial optimization problem. The only complicating feature of the problem is that classes must be grouped together into sequences known as pipelines. An ideal schedule will have all classes in a pipeline scheduled in consecutive weeks. The objective is to eliminate the nonproductive time spent by sailors at C–Schools who are waiting for the next class in a pipeline. In this investigation, five algorithms were specialized for this problem and empirically analyzed on a set of actual scheduling problems. The algorithms evaluated include simulated annealing, a greedy heuristic, tabu search, a genetic algorithm, and evolutionary programming. In empirical testing the greedy heuristic was found to be best for this application.

In addition, an implicit enumeration procedure for this problem was also developed. The implicit enumeration algorithm uses the output from the greedy algorithm as the initial upper bound and develops a lower bound based on the problem input data. If the bounds are equal, then an optimal schedule has been found; otherwise, an implicit enumeration procedure is initiated which if allowed to run long enough will eventually find an optimal schedule. On five of the six test problems, a provable optimum was obtained. The sixth problem remains unsolved.

The best ideas for Navy C–School scheduling were extended for basic training courses which are offered in Navy A–Schools. In empirical tests, we found that our heuristic algorithm worked very well on the A–School test problems.

Acknowledgment

This work was supported by the Navy Personnel Research & Development Center under the auspices of the U. S. Army Research Office Scientific Services Program administered by Battelle (Delivery Order 1110, Contract Number DAAL03–91–C–0034) and by the Office of Naval Research under Contract Number N00014–95–1–0645.

I. INTRODUCTION

The Navy currently has approximately 400,000 sailors each of whom is assigned to a job billet for a specific tour of duty lasting from two to five years. After the tour is completed, a sailor is assigned to a new billet for another tour of duty. Sailors assigned to sea billets are usually rotated to shore billets and vice versa. Each month, the Navy assigns thousands of new recruits and rotating sailors to vacant billets (jobs). The assignments are made by some 200 detailers within the Personnel Assignment Division of the Bureau of Navy Personnel. Enlisted personnel assignment is a complex process which often involves trade–offs among several conflicting policies. An excellent description of the Navy's system may be found in Blanco and Hillery [1994].

Before assuming a new assignment, a sailor frequently is required to complete one or more Navy training classes. Experienced personnel attend advanced skills training courses while new recruits take a set of basic courses. The advanced courses are offered by Navy C–Schools and the basic courses are offered by Navy A–Schools. Multiple offerings of over 2000 different courses must be taught each year with a course lasting from one week to over six months. The Navy's annual budget for training exceeds $1.3 billion.

Based on the forecast demand for courses, the C–School and A–School managers develop a schedule of class offerings for use by the detailers. Until recently these schedules were prepared manually by personnel at each of the Schools. The creation of an optimal schedule for either a C–School or an A–School is a difficult combinatorial optimization problem. The objective of this study is to develop and empirically evaluate algorithms for creating good class schedules for both types of Navy Schools.

II. NAVY C–SCHOOLS

The assignment to a new billet may require that the sailor update his/her skills which is accomplished by taking courses in the Skill Progression Training Program. These courses are taught at several campuses in the continental U.S.A. with the largest campuses being at the Fleet Training Center in Norfolk and at the Service School Command Headquarters in San Diego. The organizations which offer these courses are called C–Schools. During FY95 these C–Schools offered several thousand different courses to approximately 60,000 enlisted personnel.

As the military draw down proceeds and budgets for skills training is decreased, it is imperative that these Schools operate as efficiently as possible. An important problem faced by the management of these schools is the development of a good class schedule. In this section we describe the pipeline scheduling problem, present a set of algorithms for this problem, provide an empirical evaluation of our algorithms, and finally give our recommendation for solving this problem.

2.1 The Pipeline Scheduling Problem

The Service School Command at the San Diego Naval Station offers four courses in the area of maintenance of electronic communication equipment for the FFG–7 ship class. A sailor who successfully completes all four courses is called a *Small Combatant Communications Subsystem Technician*. The courses are as follows:

(i) HFSYS – (High Frequency Systems – 12 weeks),

(ii) WSC3 – (AN/WSC–3 Communications Sets Maintenance – 12 weeks),

(iii) NAVMACS – (Organization Level Maintenance on the Navy Modular Automated Com-

munications System – 12 weeks), and

(iv) VRC46 – (AN/VRC–46 Radio/Transmitter Set Maintenance – 2 weeks).

During 1995 this School offered HFSYS and VRC46 eight times, WSC3 nine times, and NAVMACS sixteen times. A *class*, as distinguished from a course, is a specific offering (convening date) of a course. Hence, during the one year planning horizon, there will be eight classes of courses HFSYS and VRC46, nine classes of WSC3, and sixteen classes of NAVMACS. Since a sailor can only take one class at a time, a sailor who needs all four courses will be assigned to this campus for a minimum of thirty eight weeks.

This School runs two shifts (a day shift and an evening shift) for courses HFSYS, WSC3, and NAVMACS, and a day shift only for VRC46. Many sailors come for all four courses and proceed through four classes grouped together into what is known as a pipeline or pipe. That is, a *pipe* is a sequence of classes that are scheduled so that a sailor can take all four courses before departing for the next assignment. The four courses in a pipe can be taken in any order and ideally they will be offered back–to–back.

Classes are generally team taught and there may be several teams available to teach the same course. When there are multiple teams, they teach during different shifts (either the day shift, the evening shift, or the night shift). Hence, WSC3 shift 1 and WSC3 shift 2 refer to the same course (WSC3) being taught by different teams at different times of the day. For this study the terms team and shift will be used interchangeably.

Since many courses have multiple instructors, it is sometimes possible for a given team to begin a new class before a previous class is finished. The minimum time required between the starting times of two classes is called the *offset*. For example, suppose course A lasting 10 weeks has a two week offset. Then classes could begin in weeks one, three, five, seven, et cetera. This partial schedule is illustrated in Figure 1.

Figure 1. About Here

Ideally the classes in a pipe will be offered back–to–back so that a sailor has no idle weeks at the C–School. A period of one or more weeks in a pipe in which no class is offered is called a *gap* in the pipe. Pipe 1 illustrated in Figure 2 has no gaps while pipe 2 has gaps totaling four weeks. The pipes in a perfect schedule will have no gaps.

Figure 2. About Here

These ideas are further illustrated by examining the problem data for the FFG–7 problem presented in Table 1. Line 1 indicates that this schedule is for the fiscal year 1995 (1 October 1994 – 30 September 1995), and is a one year schedule. A one year schedule corresponds to 50 weeks with a two week break for Christmas. A two year schedule corresponds to 100 weeks and all schedules are for at most two years.

Table 1. About Here

Line 3 indicates that the schedule is composed of eight pipes each of which has four courses. The names of the four courses are given on lines 4, 8, 12, and 16. Course A lasts 12 weeks and there are two teams (shifts) available to teach it. The first shift will offer this course four times during the year with an offset equal to the course length. That is, the second offering of course A using shift 1 cannot begin until the completion of the first offering. For course B, shift 1 will offer five classes during the year while shift 2 will only offer four. This gives a total of nine class offerings for the eight pipes. Hence, one of the classes will not appear in any pipe. An offset of eight for course B shift 1 indicates that if a class begins in week t another class could begin in week t+8. Hence, in weeks t+8, t+9, t+10, and t+11 there would be two classes being

taught by shift 1. For course C there will be a total of 16 classes of which only eight will appear in pipes.

Based upon its characteristics, a schedule is assigned a nonnegative score. Penalty points are assessed for undesirable characteristics with a perfect schedule having a score of zero. Gaps in pipes result in penalty points equal to the total number of weeks in the gaps. A class which extends beyond the planning period is assessed penalty points equal to the number of weeks that it extends beyond the planning horizon. For example a class that was completed at the end of week 56 when the planning horizon is 50 weeks would be assessed a penalty of six points. Each pipe is assigned a minimum target week for its start date. A pipe is assessed one penalty point for each week of time that it begins prior to its minimum target week. Pipes may begin on or after the target week with no penalty.

The minimum pipeline target for pipe p (E_p) is based on the number of weeks in the planning horizon (W) and the number of pipes (P) in the schedule. This relationship is $E_p = \lfloor (p-1)W/(2P) \rfloor + 1$. For FFG−7, the pipeline targets for each of the eight pipelines are 1, 4, 7, 10, 13, 16, 19, and 22. A feasible schedule having a score of 19 appears in Figure 3. The eight pipes are denoted by the lower case letters a, b, ...,h. Pipe a consists of the classes A1a, D1c, C1d, B1d. The 0's in Figure 3 refer to classes which do not appear in any pipe. The detailed calculations of this score may be found in Tables 2 and 3. The six, three, and ten in Table 2 are due to the three classes which extend beyond the 50 week planning horizon.

Figure 3, Table 2, and Table 3. About Here

2.2 Heuristic Algorithms for the Pipeline Scheduling Problem

Heuristic procedures for discrete optimization problems are usually based upon one of three ideas: (i) greedy heuristics, (ii) interchange heuristics, or (iii) truncated exponential procedures. This section presents, a greedy heuristic and several interchange heuristics which

have been specialized for the pipeline scheduling problem. The interchange heuristics include simulated annealing, tabu search, a genetic algorithm, and evolutionary programming. Some general information about all of these techniques can be found in Parker and Rardin [1988], and Nemhauser and Wolsey [1988, 1989].

2.2.1 Simulated Annealing

The term simulated annealing comes from the thermodynamics analogy with the way that metals cool or anneal (see Press, Flannery, Teukolsky, and Vetterling [1989]). If a liquid metal is cooled slowly, then the atoms form a crystal which has a minimum energy state. If a metal is cooled too quickly or quenched, then it may end up in a polycrystalline state having a higher energy level. The idea behind simulated annealing is to use slow cooling in an attempt to obtain a low energy state (objective value).

A software implementation of the simulated annealing algorithm specialized for the pipeline scheduling problem is called Pipeline Assistant. It begins with a set of equally spaced start times for each class and a random assignment of classes to pipes. This starting schedule may violate both the offset restrictions and the restriction that a sailor cannot be in two classes simultaneously. A very large penalty is assessed for these constraint violations. At each iteration a neighboring schedule is generated at random. Generating a neighbor will involve either moving a single class one week earlier or one week later, or switching a pair of classes of the same course between two pipes.

If the neighbor has a lower score than the current schedule, then the neighbor becomes the current schedule and the process is repeated. If the neighbor's score is worse than that of the current schedule, then a random number is used to determine if the neighbor becomes the current schedule. Suppose the current schedule has a score of S_c and the neighbor has a score of S_n with $S_c < S_n$. Let R be a random number from a Uniform [0,1] distribution, and let T be a parameter known as the temperature. If $R < [1 - (S_n - S_c) / T]$, then the neighbor becomes the new current schedule. Initially T is set to 5% of the score of the initial schedule. As the

iterations proceed, the temperature is systematically lowered by replacing T with 0.8T. The procedure searches for some finite number of iterations and retains the best schedule found during the search. In Pipeline Assistant this is repeated three times and these three schedules are reported to the user. There is no guarantee that any of these schedules will be feasible.

2.2.2 A Greedy Heuristic With Local Improvements

The greedy heuristic develops a schedule similar to the way a human would construct a schedule manually. Initially the courses are given a priority ranking. For a four course pipeline, course A is given priority 1, course B is given priority 2, and so forth. The pipes all begin at their minimum target times and are constructed one class at a time. At each iteration, the shortest incomplete pipe is selected for a class assignment. The first available class associated with the unassigned course having highest priority is assigned to this pipe at the earliest possible time. This strategy attempts to construct pipes having no gaps. After all pipes have been constructed, the other classes are scheduled one at a time at the earliest time possible. At this stage a feasible schedule has been developed.

Each pipe having a gap of at least one week is examined in an attempt to start the pipe later and thus reduce the gap. Finally, for those pipes in which the last class begins after the planning horizon, an attempt is made to move this class to the first position in the pipe. Fewer penalty points will be incurred for beginning this pipe prior to its target week than are currently being assessed for beginning after the planning horizon. This results in one candidate schedule.

Other candidate schedules are obtained by changing the course priorities. The greedy algorithm tries a large number of possible priorities and saves the best schedule found. For a problem having n courses, min(10K, n!) possible schedules will be developed. Some theoretical results concerning the optimality of greedy algorithms on ordering problems may be found in Dechter and Dechter [1989].

2.2.3 The Tabu Search Algorithm

Tabu search is a heuristic technique that is based upon selected concepts from the field of artificial intelligence. It is a flexible method for guiding a search over the space of solutions in an attempt to discover a high quality solution. Tabu search uses three types of rules that attempt to prevent the search from being trapped at an inferior local optimum.

The first set of rules guides the local search. Let w be a given number of weeks that a class is allowed to move, either forward or backward. For a given schedule x and a given class c, the neighborhood N(x,c), is defined as the set of all schedules formed from x by moving c at most w weeks. A neighborhood N(x,c) is said to be tabu if class c has been moved during the most recent q iterations. The systematic use of this rule provides the tabu search with a short term memory that prevents the local search from revisiting the same neighborhood. The "most improving" selection rule would select the best move over all N(x,c) for all classes c. For this application the most improving rule is numerically expensive and requires the evaluation of each move generated by each (course, shift, class, week) combination. In our tabu search procedure, we use a hybrid of the most improving rule and a Monte Carlo method. Specifically, the best move from h randomly generated neighbors is selected.

The second set of rules diversify the search, by moving the local search into a new region of the solution space. For a given schedule x, the new region is reached by swapping a pair of classes of the same course between two pipes. This triple (pipe 1, pipe 2, course) is the best swap among m randomly generated candidates. To prevent the search from cycling back to the same schedule x, once a swap is executed it will be tabu for the next n iterations. This is usually referred to as intermediate memory.

The third set of rules define the aspiration criteria, which may be viewed as a relaxation of the first two rules. Since the potential moves using the above rules are generated randomly, it is possible that a tabu move will be produced. If the resulting score is an improvement over the best schedule found so far, then this tabu move is accepted.

Given a schedule x, the procedure applies the first set of rules for f iterations. This is followed by one application of the second rule (a pipe swap). Since an infeasible solution may be developed, as with simulated annealing, a very large penalty is assessed for violating feasibility restrictions. We begin the tabu search with the first feasible solution developed by the greedy heuristic. Additional information about this general approach may be found in Glover [1989, 1990a, 1990b] and Glover and Laguna [1993].

2.2.4 The Genetic Algorithm

The genetic algorithm begins with a schedule generated by the initial application of the greedy heuristic. At the start of the algorithm, this schedule is the lone member of a pool of schedules that will be built up and modified from iteration to iteration. Each schedule in the pool is evaluated to determine how well it satisfies the problem requirements and is assigned a score based upon that evaluation.

The algorithm uses two basic operations, *mutation* and *cross−over*. Given a schedule, a mutation replaces a random number of randomly selected classes with randomly selected classes of the corresponding course and shift. A cross−over is slightly more complicated and involves combining two schedules. Given two schedules, randomly select two points for interchange. The two points are selected so that they occur at pipeline boundaries. Place pipes in the first schedule prior to the first interchange point and pipes in the second schedule between the first interchange point and the second interchange point into the *result* schedule. Then fill out the result schedule with pipes (if any) and other classes from the first schedule after the second interchange point.

An iteration of the genetic algorithm proceeds as follows:

1. Randomly select a given proportion of the pool to be subjected to the mutation operation. (Selected schedules remain in the pool.) Add the schedule resulting from the mutation operation to the pool. The random selection of schedules to mutate is biased according to the schedule score. Better schedules are more likely to be mutated.

2. Randomly select a given proportion of the pool to be subjected to the cross–over operation. (As with mutation, selected schedules remain in the pool.) Add the schedule resulting from the cross–over operation to the pool. The random selection of schedules to cross–over is also biased according to the schedule score. Better schedules are more likely to be selected for the cross–over operation.
3. Reduce the pool to its maximum allowable size by randomly removing schedules. (This is not necessary in the first few iterations.) The random selection of schedules to remove is biased according to the schedule score. Better schedules are less likely to be removed, and the best schedule is never removed.

In any particular iteration, schedules may be generated that violate one or more of the problem restrictions. In addition to those stated in the section on simulated annealing (the offset and the class overlap restrictions), the limit on the number of classes of a particular course and shift may be violated. Very large penalties for these constraint violations are added to schedule scores of the infeasible schedules. These infeasible schedules are added to the pool when they are created; however, they are much more likely to be removed from the pool when its size is reduced at the end of each iteration.

The algorithm repeats the process for a specified number of iterations and then reports the best schedule found. Additional information about this general approach may be found in Winston [1992].

2.2.5 The Evolutionary Programming Algorithm

Evolutionary programming is a procedure developed by D. B. Fogel of ORINCON Corporation which also uses ideas from biology to produce heuristic algorithms for a variety of discrete optimization problems. For example, an algorithm for the traveling salesman problem can be found in Fogel [1988]; a technique for solving linear systems may be found in Fogel and Atmar [1990]; and a technique for training neural networks can be found in Fogel, Fogel, and Porto [1990]. J. P. Lemoine [1994] adapted evolutionary programming for the pipeline scheduling problem and developed a software implementation.

Using the greedy algorithm to obtain starting schedules, Lemoine develops an initial set of parent schedules which are then perturbed by adding random gaps. The scores for each parent schedule are used in a competition based upon random numbers, that determines which parents survive to become the basis for the next generation. The survivors create progeny (a new generation of schedules) and the process repeats. The progeny are produced using two methods. One routine randomly adds and removes gaps to a schedule. The other randomly switches positions of two classes in a randomly selected pipe.

For this application, a generation begins with 20 schedules. A competition reduces these 20 schedules down to four survivors. Each of these four survivors produces five progeny resulting in a new generation of 20 schedules. Lemoine found that good solutions could generally be obtained in about ten generations.

2.2.6 Empirical Analysis

Six actual problems from various C–Schools are used in the empirical tests. The most difficult problem (PAS_2) has ten courses and eight pipelines (see Table 4). All problems have a one year planning horizon (50 weeks) and no course is taught by more than three shifts. The course length varies from one to 31 weeks.

Since Pipeline Assistant is written in Turbo Pascal and uses routines from Microsoft Windows, the first test was conducted on a 486–based PC. A comparison of Pipeline Assistant with a C language implementation of the greedy algorithm may be found in Table 4. The times are wall clock times and are rounded to the nearest second. For every problem, the greedy implementation produced a better solution in less time. Since Pipeline Assistant uses a random number seed based upon the system clock, additional runs yield different solutions. In five additional runs with FFG–7, Pipeline Assistant produced schedules having the following scores: 27, 34, 28, 26, and 20.

Table 4. About Here

An empirical analysis of C language implementations of the other algorithms may be found in Table 5. Reported times are *user times* (estimates cpu time) on a DecStation 5000/240. For each of the interchange heuristics, the initial solution, the final solution, and the total time is reported. Both the tabu search and the genetic algorithm apply an elementary version of the greedy method to obtain initial schedules. The evolutionary programming algorithm applies a more advanced version of the greedy algorithm to obtain the initial schedule. The greedy algorithm produced the best schedule for each of the six problems and was always faster than tabu search and the genetic algorithm. The evolutionary programming code ran each problem in approximately 17 seconds regardless of the problem size. On PAS_2 the evolutionary programming implementation ran faster than the greedy implementation, but failed to produce as good a solution.

The evolutionary programming software begins by generating 20 schedules using the greedy method with local improvement. This is followed by 15 iterations of the evolutionary programming algorithm. The score of the best of the greedy schedules is reported in the row entitled initial score. Notice that the computational machinery of evolutionary programming never discovered a schedule that was superior to that produced by the 20 trials of the greedy algorithm. Hence, the best solution obtained by the evolutionary programming software was obtained in the first second by the greedy heuristic.

Table 5. About Here

2.3 An Implicit Enumeration Algorithm for the Pipeline Scheduling Problem

In this Section we present an enumeration algorithm for the pipeline scheduling problem. A lower bound is developed from an analysis of the input data and feasible schedules (upper bounds) are generated one pipe at a time using a specialized enumeration strategy. Hence, a partial schedule whose score is greater than or equal to the incumbent (best schedule found so far) can be fathomed (discarded). The hope is that a good incumbent will permit the algorithm to fathom most of the possible schedules.

2.3.1 The Lower Bound

Suppose there are P pipes and suppose the minimum duration of each of these pipes is D weeks. That is, if a pipe has all classes scheduled back−to−back, then the pipe lasts D weeks. Let T_p denote the target start date for pipe p. If pipe p begins on or after week T_p there is no penalty. Let H denote the planning horizon (either 50 or 100 weeks). If a pipe lasts through week H+t, then the score for this pipe is at least t. Therefore, $L = \max[0, H - (T_1 + D - 1)] + \max[0, H - (T_2 + D - 1)] + \ldots + \max[0, H - (T_P + D - 1)]$ is a lower bound for any feasible schedule.

2.3.2 Complete Enumeration

In order to develop an implicit enumeration algorithm, one needs a technique to completely enumerate all possible solutions since in the worst case, the algorithm will require a complete enumeration of the solution space. Consider a problem having three courses A, B, C and a single pipe. There are 3! = 6 possible schedules given by the permutations (ABC, ACB, BAC, BCA, CAB, CBA). That is, given a permutation, convening dates for the classes are determined by the earliest possible time for each class in the permutation. For the permutation BAC, class B is convened at the earliest possible week, the target week for the pipe. Class A begins as soon as B is completed followed by C. If the

schedule has three courses and two pipes, then there are (3!)(3!) = 36 possible schedules as illustrated in Figure 4. In general for a schedule having C courses and P pipes there will be $(C!)^P$ possible schedules formed in this way.

Figure 4. About Here

Note that the tree in Figure 4 has 6+36=42 nodes. The first six nodes represent the first pipe and corresponding priorities for the three courses, whereas the next level of nodes represent the second pipe and corresponding priorities for its courses. In general a tree will have $(C!) + (C!)^2 + (C!)^3 + ... + (C!)^P$ nodes. A very compact algorithm based upon recursion has been developed to generate these nodes. A node is defined by the pair (Z,p) where p denotes the pipe number and Z is a P by C matrix where Z_{pi} denotes the course in the ith position of pipe p. The complete enumeration algorithm is given as follows:

algorithm complete enumeration(P, C)

Inputs: P – denotes the number of pipes

C – denotes the number of courses

Output: A set of $(C!)^P$ matrices Z corresponding to all of the permutations

begin

for p = 1 to P **do** **for** i = 1 to C **do** $Z_{pi} \leftarrow i$;

perm(1, 1, Z);

end

```
procedure perm(p, i, Z)
Inputs:  p − denotes the current pipe
         i − denotes the current course position
Output: A set of Z matrices
begin
   if i < C then
      if i < P then  perm(p+1, 1, Z);
      else  write Z;
   else
      for j = i to C do
         switch(p, i, j, Z);
         perm(p, i, Z);
         switch(p, i, j, Z);
      end
   end
end

procedure switch(p, i, j, Z)
begin
   t ← Z_pj;    Z_pj ← Z_pi;    Z_pi ← t;
end
```

The P row C column matrix Z will correspond to one permutation of the P pipes. Hence, the first six permutations illustrated in Figure 4 correspond to

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}.$$

2.3.3 Construction of a Pipe

For a pipe with a given permutation (i. e. Z), the convening dates for the classes in the pipe are determined sequentially. Given a partial schedule (i. e., some classes have been scheduled) and a new class to be scheduled, we need an efficient method to determine the earliest possible convening date for this new class. Since the C−School courses are generally team taught, they frequently have multiple offerings of the same course being offered simultaneously, as long as the starting dates maintain the offset. Suppose course A having a two week offset currently has three offerings scheduled as illustrated in Figure 5. Then weeks 8, 12, 15, and 20 are convening dates which we wish to consider. That is, we consider the earliest possible starting time and the convening times plus the offset for the other three classes. Let γ denote a possible trial value for class 4. If γ is in the open interval (10−2,10+2), then γ will violate the offset restriction for class 1. Likewise, if γ is in the open interval (13−2, 13+2), then γ will violate the offset restriction for class 2. If A_i is the convening time for class i with an offset of α, then γ can not be in the open interval $(A_i - \alpha, A_i + \alpha)$ for any i. These ideas result in the following assignment algorithm:

algorithm assign(n, A, α, β, γ)

Inputs: n − denotes the number of currently scheduled classes for this course

A_i − denotes the convening week for class i

α − denotes the offset

β − denotes an earliest possible convening week

Output: γ − convening week for class n+1

Assumption: $A_1 \leq A_2 \leq ... \leq A_n$

begin

$\gamma \leftarrow \beta$;

for i = 1 to n **do** **if** $A_i - \alpha < \gamma < A_i + \alpha$ **then** $\gamma \leftarrow \max(\gamma, A_i + \alpha)$;

end

Given a permutation of pipes and classes defined by Z, assign can be used to construct a schedule. That is, assign determines the convening time for each class in Z.

2.3.4 Implicit Enumeration Algorithm

Given a permutation matrix Z and a $p < P$, a partial schedule can be developed by applying assign sequentially to the first p pipes. Let score(Z,p) denote the score of the partial schedule defined by Z and p. Since score(Z,p) $\leq$ score(Z,p+1) then score(Z,p) is a lower bound on score(Z,P). Combining this notion with the complete enumeration algorithm results in the following implicit enumeration algorithm:

algorithm implicit enumeration(P, C)

Inputs: P – denotes the number of pipes

C – denotes the number of courses

Output: v – denotes the score of the best solution found

begin

for p = 1 to P **do** **for** i = 1 to C **do** $Z_{pi} \leftarrow i$;

let v denote the score of the solution found using the greedy heuristic;

let L denote a lower bound based on the analysis given in Section 2.3.1;

if v > L **then** enumerate(1, 1, Z, v);

write "the best schedule found has a score of v"

end

```
procedure enumerate(p, i, Z, v)
Inputs:  p – denotes the current pipe
         i – denotes the current course position
         Z – denotes a permutation
         v – denotes the current best solution
begin
   if i < C then
      if p < P then
         if score(Z,p) < v then enumerate(p+1, 1, Z, v);
      else if score(Z,P) < v then v ← score(Z,P);
      end
   else
      for j = i to C do
         switch(p, i, j, Z);
         enumerate(p, i, Z, v);
         switch(p, i, j, Z);
      end
   end
end
```

Of course, one must also save the schedule whenever v is updated. The hope is that the initial solution (upper bound) will be small and that the lower bounds will be large.

2.3.5 Empirical Analysis

The six test problems were run using a C language implementation on a 275 MHz Alpha based DecStation and the results are summarized in Table 6. For the first four problems, the greedy heuristic found a solution whose value was equal to the lower bound; hence, no enumeration was needed. A provable optimum for the FFG–7 problem was obtained after approxi-

mately one−half hour of computer time. On PAS_2 the system was terminated after it examined three million nodes which took approximately one hour of cpu time. At termination, no improvement over the greedy heuristic had been made and the gap between the upper and lower bounds was approximately 25%.

Table 6. About Here

2.3.6 Recommendation

Based on our empirical analysis, we recommend an algorithm that uses three procedures: (i) the greedy heuristic, (ii) the lower bound, and (iii) the implicit enumeration algorithm. The greedy heuristic has consistently produced fairly good schedules which could be used by the client. The lower bound should also be produced and reported to the user. If the gap between the upper and lower bound is large, then the user should be given the option to run the implicit enumeration algorithm for some given time, to determine if the gap can be closed. A finite termination criteria should be incorporated to prevent excessive run time in the implicit enumeration algorithm.

III. NAVY A–SCHOOLS

The basic courses for new recruits are offered at campuses known as A–Schools. The largest are located at Norfolk, San Diego, and Great Lakes, and during FY96 over 80,000 sailors took at least one A–School course. The managers of each A–School is responsible for planning the class schedule for their School. In this Section we describe the A–School class scheduling problem, we present a greedy heuristic for this problem, and we provide an empirical evaluation of our algorithm.

3.1 Description of the Problem

A Navy A–School will offer a variety of training courses during a one year planning period. Some courses last only a few days while others may involve over one hundred days of instruction. Some days a class may meet in a large lecture hall and other days the class may be divided into smaller groups. These smaller groups will meet in different rooms with individual instructors. Hence, on day one, the School may need a large lecture hall and one instructor but on day two, the School may need three small lecture rooms and three instructors. The instructors are interchangeable so that an instructor can teach either the whole group in a large lecture hall or the smaller groups. In addition, some classes may require special equipment such as calculators or personal computers. The managers of the A–Schools are responsible for developing a class schedule that requires the minimum number of instructors subject to resource constraints on lecture halls, seminar rooms, laboratory space, and equipment.

The input data used to describe the A–School class scheduling problem is given below:

Let i – denote a course (i= 1,...,n)

Let D_i – denote the duration of course i in days. (This can vary from a few days to well over one hundred days.)

Let O_i – denote the number of times course i will be offered during the year. All offerings

could convene on the same day or they could be dispersed throughout the year. Dispersal usually results in the requirement of fewer instructors on the busiest day.

Let I_{id} – denote the number of instructors needed by course i on day d. This can vary from a minimum of one to a maximum of four.

Let C_{jid} – denote the consumption of resource j by course i on day d. These are usually 0 or 1 depending upon whether a particular type room is needed.

Let R_{jt} – denote the number of resources of type j available on calendar day t. Usually the consumption is a small integer corresponding to the number of rooms of a given type available at the School.

The decision variables are convening dates for the $O_1 + O_2 + ... + O_n$ classes to be offered. The objective is to select convening dates that require the least number of instructors on the busiest day of the year. The problem may be further complicated by the imposition of convening day restrictions. For example, some A–Schools require that a certain course always begin on a Monday. Some may restrict the number of offerings of a given course in a given month.

3.2 The Greedy Algorithm

Consider an A–School which offers $O_1 + O_2 + ... + O_n = m$ classes during the year. Suppose this School is open Monday through Friday except for thirteen holidays. For a non–leap year, this results in $356 - 104 - 13 = 248$ possible convening days for each of m classes. Hence, there are 248^m possible schedules.

Based upon the successful implementation of the greedy algorithm for C–Schools, we developed a similar approach for the A–School scheduling problem. Consider a School that offers three courses, A, B, and C. We first develop the 6 permutations for these courses (ABC, ACB, BAC, BCA, CAB, CBA). Suppose that A is offered twice, B once, and C three times. Using a simple rule to spread the offerings of each course throughout the year, we obtain the following possible sequences:

Sequence Number	Class Order
1	A1–B1–C1–C2–A2–C3
2	A1–A2–C1–C2–B1–C3
3	B1–A1–C1–A2–C2–C3
4	B1–C1–A1–C2–A2–C3
5	C1–A1–B1–C2–A2–C3
6	C1–B1–A1–C2–A2–C3

For sequence 1, the greedy algorithm will schedule the first offering of A followed by the first offering of B. This is followed by two offerings of course C, the second offering of A, and the final offering of C.

The greedy algorithm initializes the day index t to 1 and assigns the first class in the sequence a convening day of t. For sequence 1, A1 convenes at day 1. Next it attempts to assign the second class in the sequence a convening day of t. If a resource violation occurs, then t is incremented by one and it tries again. Once the second class is scheduled, then it proceeds to the third. This process continues until either all classes are scheduled within the year (a feasible schedule) or it fails to find a feasible schedule. If a feasible schedule is obtained, then an upper bound on the number of instructors needed has been established. Suppose this process results in a feasible schedule using 50 instructors on the busiest day. This first schedule found does not attempt to minimize the number of instructors and the algorithm is concerned only with feasibility. In the next stage the number of instructors is systematically reduced using a binary search on the interval [1,50] and the algorithm proceeds to find a feasible schedule with this sequence and the minimum number of instructors. This strategy is repeated with the rest of the sequences and the best of the best is saved.

3.2 Empirical Analysis

Seven problems were used in our empirical tests. The number of courses varies from one to six and the number of resource restrictions varies from one to nine. The course length varies from ten days to over one hundred days and the number of offerings of a single course during the year varies from one to sixty. The characteristics of these test problems may be found in Table 7.

Table 7. About Here

SABRE Decision Technologies has developed a PC based system to obtain schedules for this problem. The system is called NCSS (Navy Class Scheduling System) and uses simulated annealing in an attempt to obtain a good schedule. Table 8 summarizes our results using both simulated annealing and the greedy heuristic on the seven test problems. On six of the problems , greedy provided a better answer and on the seventh there was a tie. Since NCSS is written in Visual Basic, it was run on a 50 MHz 486 based PC and each run took approximately 15 minutes. The greedy code is written in C and those runs were made on a 275 MHz Alpha based DecStation and never required more than a few seconds.. These results are consistent with those obtained previously with the pipeline scheduling problem.

Table 8. About Here

IV. SUMMARY AND CONCLUSIONS

Five computer codes designed to obtain good solutions to the Navy's C–School scheduling problem involving pipelines have been tested. The five codes are based upon the following algorithms:

(i) simulated annealing,

(ii) a greedy heuristic,

(iii) tabu search,

(iv) a genetic algorithm, and

(v) evolutionary programming.

In empirical tests with these five codes on six actual C–School scheduling problems, the greedy algorithm produced the best schedules. Not only did it produce the best schedules, its computational time was least for all runs except for problem PAS_2. For this problem, the evolutionary programming code terminated after 18 seconds while the greedy algorithm ran for about one minute.

The four interchange heuristics (simulated annealing, tabu search, the genetic algorithm, and evolutionary programming) all involve several tuning parameters (initial temperature, iterations before a temperature change, number of iterations before the local search is abandoned and a new region is explored, the mutation and cross–over rules used, the number of schedules retained in a generation, the number of generations computed, et cetera). These tuning parameters were set by the authors of each code and were not modified in our experiments. However, it is well known that modification of the tuning parameters in any of these codes may produce different results. The only tuning parameter in the greedy algorithm is the maximum number of schedules to be developed before termination. This parameter which was set to 10,000 in these runs could probably be set to a much smaller value with no deleteri-

ous effect on the final solution. In the empirical analysis, the best solution for the problem PAS_2 was discovered at iteration 1746. It is also possible that all of the interchange heuristics could benefit by first running the greedy heuristic to completion followed by the interchange search.

We also developed an implicit enumeration algorithm for the pipeline scheduling problem. At the expense of substantial computational time, this algorithm discovered the optimal solution for the FFG−7 problem. None of the heuristic methods were able to find this optimal solution.

The ideas used in the successful greedy heuristic for the pipeline scheduling problem were extended for the A−School scheduling problem. In empirical tests, the greedy heuristic performed better than an implementation of simulated annealing on a set of test problems. Based on these studies, we conclude that greedy type algorithms are ideally suited for class scheduling problem for Navy training schools.

V. REFERENCES

Blanco, T. A. and R. C. Hillery [1994], "A Sea Story: Implementing the Navy's Personnel Assignment System," Operations Research, 42, 5, 814–822.

Dechter, A. and R. Dechter [1989], "On the Greedy Solution of Ordering Problems," ORSA Journal on Computing, 1, 3, 181–189.

Fogel, D. B. [1988], "An Evolutionary Approach to the Traveling Salesman Problem," Biological Cybernetics, 60, 139–144.

Fogel, D. B. and J. W. Atmar [1990], "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems," Biological Cybernetics, 63, 111–114.

Fogel, D. B., L. J. Fogel, and W. W. Porto [1990], "Evolving Neural Networks," Biological Cybernetics, 63, 487–493.

Glover, F. [1989], "Tabu Search, Part I," ORSA Journal on Computing, 1, 3, 190–206.

Glover, F. [1990a], "Tabu Search, Part II," ORSA Journal on Computing, 2, 1, 4–32.

Glover, F. [1990b], "Tabu Search, A Tutorial," Interfaces, 20, 4, 74–94.

Glover, F. and M. Laguna [1993], "Chapter 3 Tabu Search," in Modern Heuristic Techniques for Combinatorial Problems, Editor Collin Reeves, John Wiley & Sons, Inc., New York.

Lemoine, J. P. [1994], "Application of Evolutionary Programming to the Pipeline Scheduling Problem," unnumbered technical report, Navy Personnel Research and Development Center, San Diego, CA.

Nemhauser, G. L. and L. A. Wolsey [1988], Integer and Combinatorial Optimization, John Wiley and Sons, New York.

Nemhauser, G. L. and L. A. Wolsey [1989], "Chapter VI Integer Programming," in Handbooks in Operations Research and Management Science: Volume 1 Optimization, Editors G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, North–Holland, New York.

Parker, R. G. and R. L. Rardin [1988], Discrete Optimization, Academic Press, New York.

Press, W. H., B. P. Flannery, S. Teukolsky, and W. T. Vetterling [1989], Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, New York.

Winston, P. H. [1992], Artificial Intelligence, Third Edition, Addison–Wesley, Reading, MA.

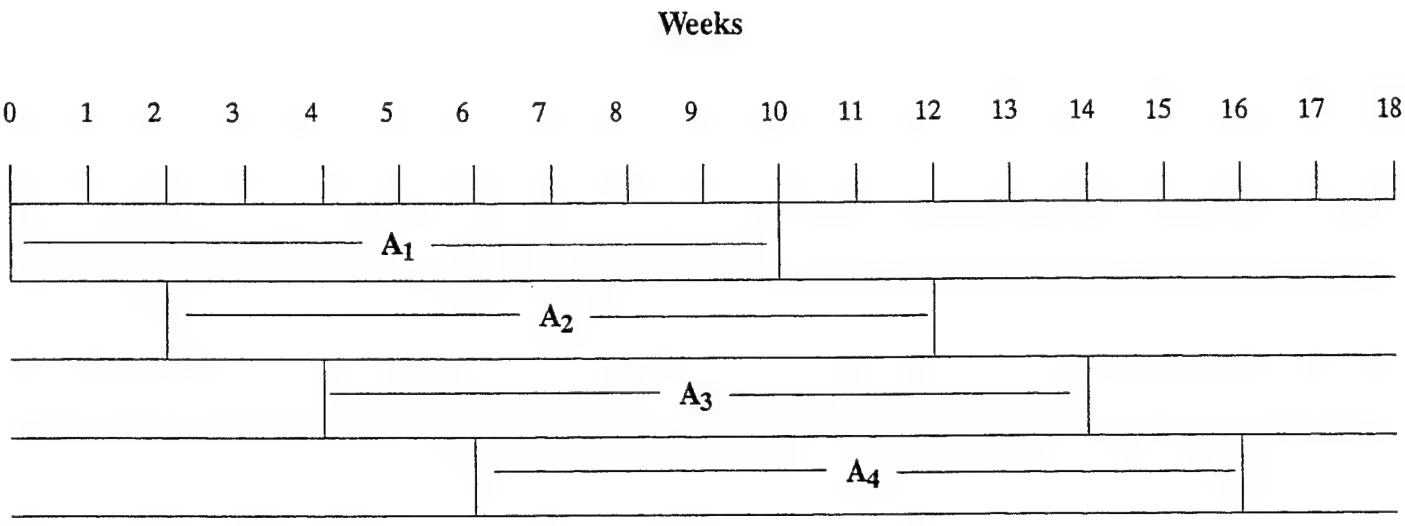


Figure 1. Schedule for a Ten Week Course Having a Two Week Offset

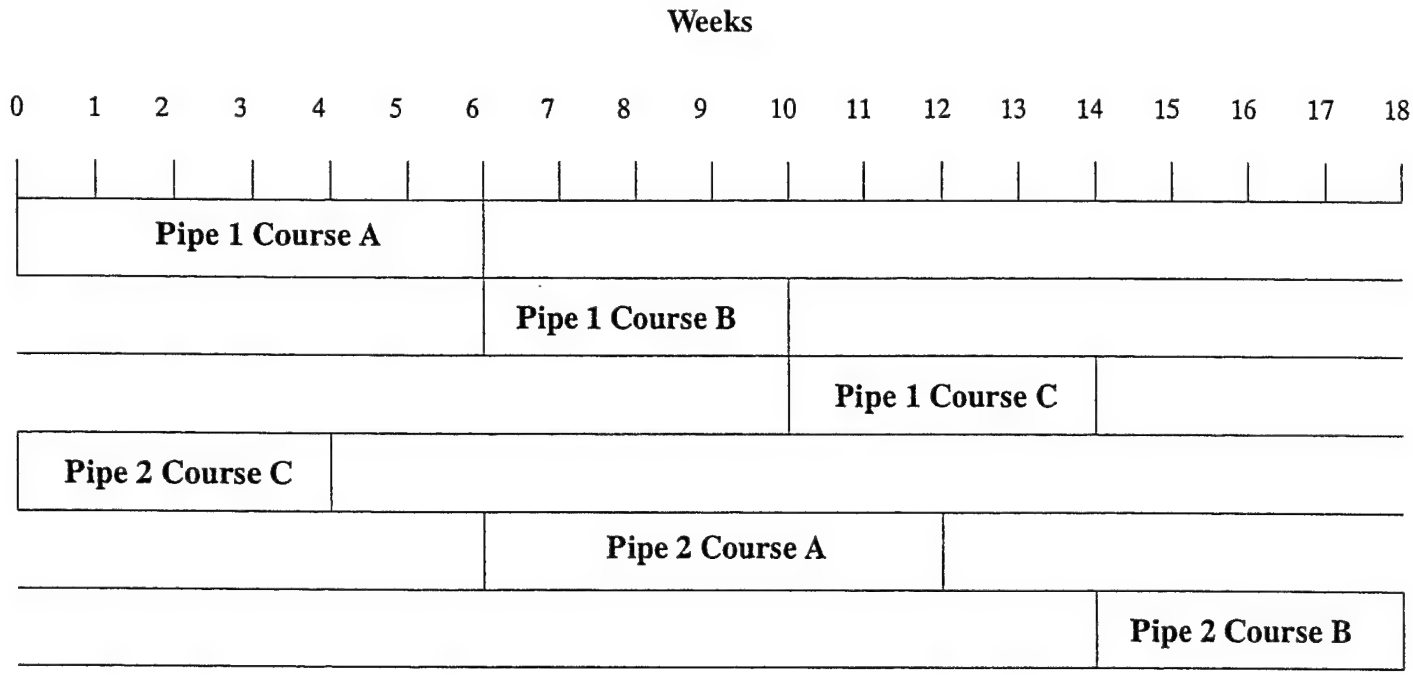


Figure 2. Example of Two Pipes
(Pipe 1 = [ABC] has no gaps)
(Pipe 2 = [CAB] has four weeks of gaps)

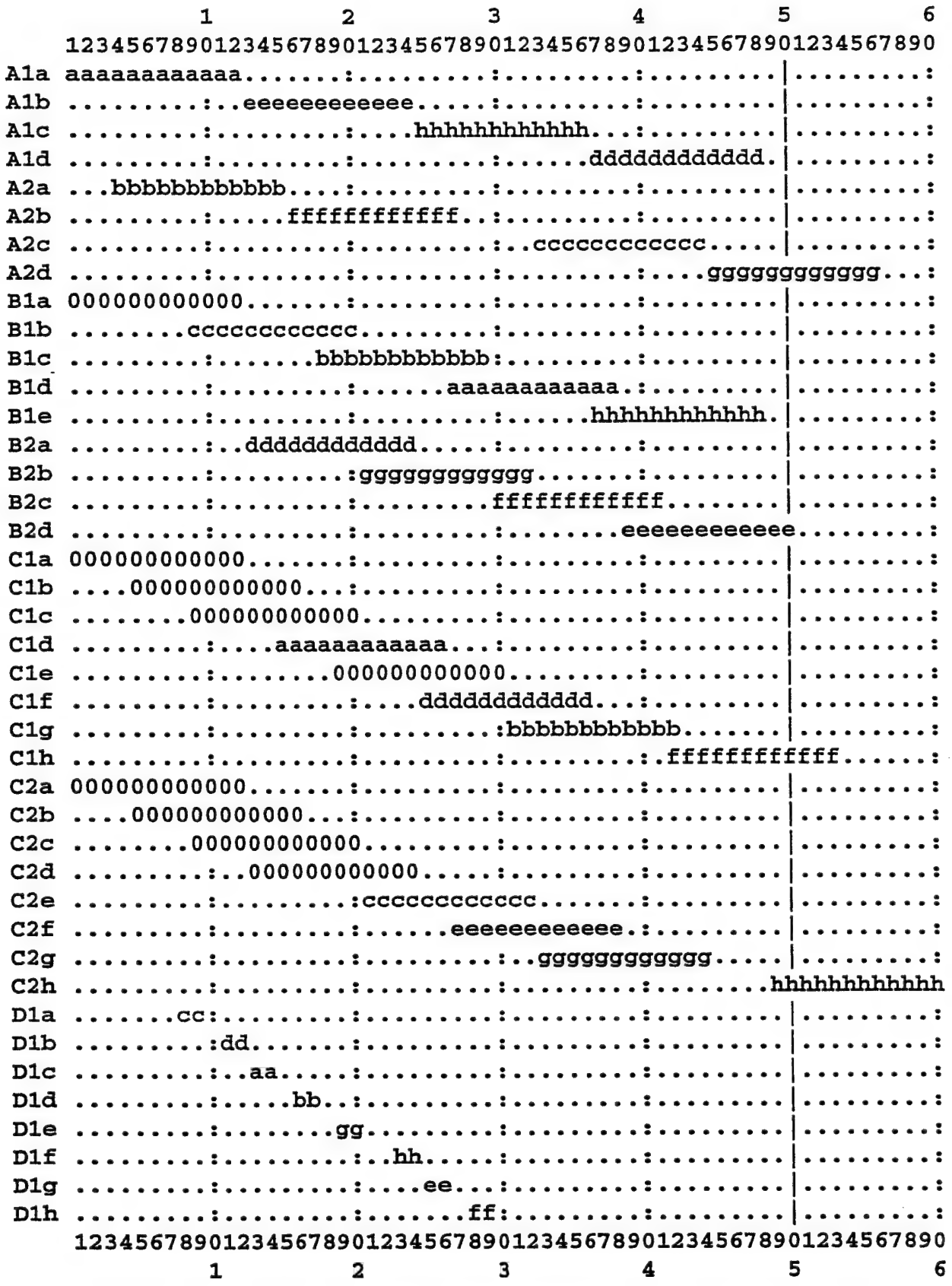


Figure 3. A Feasible Schedule for FFG7

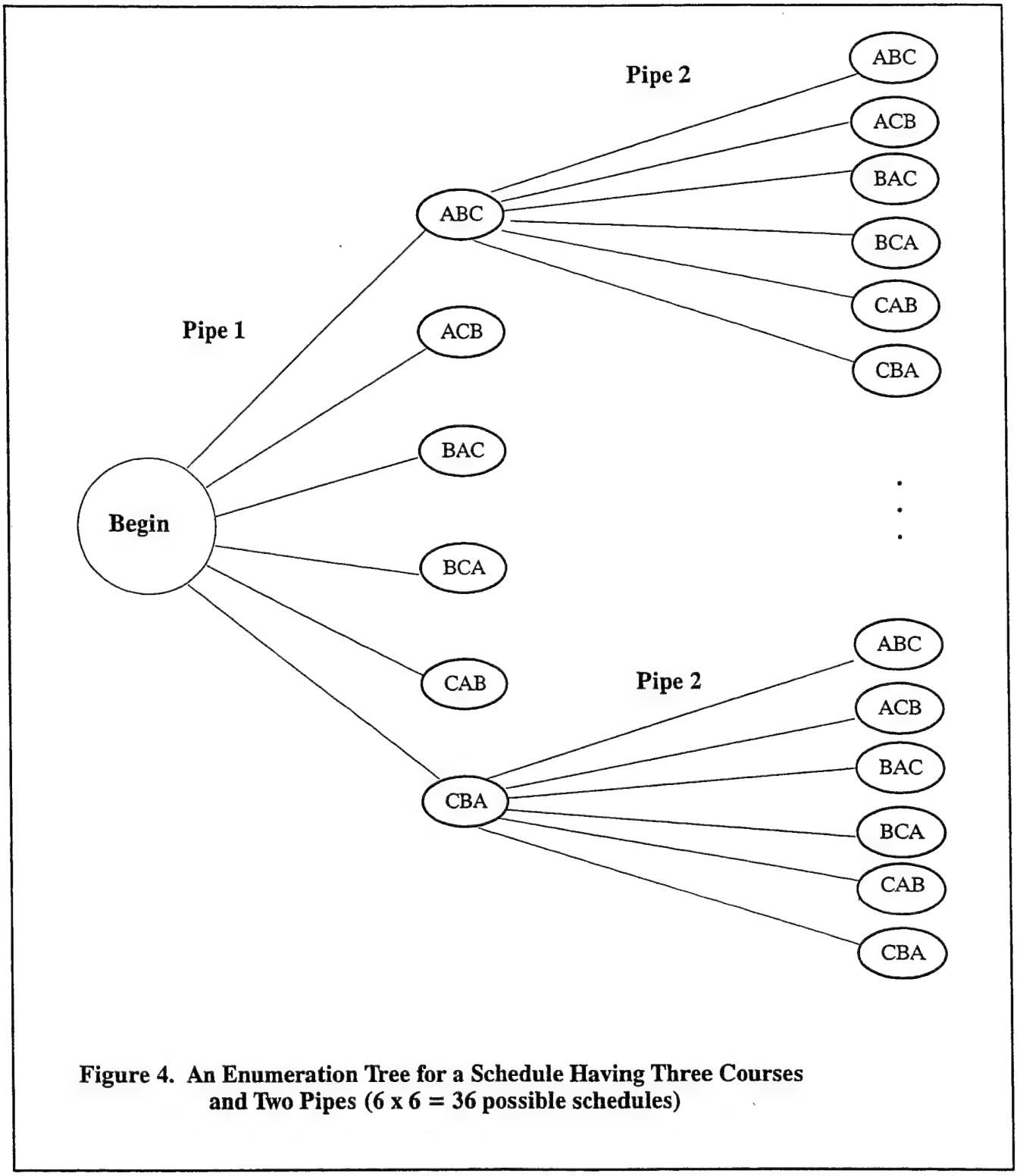


Figure 4. An Enumeration Tree for a Schedule Having Three Courses and Two Pipes (6 x 6 = 36 possible schedules)

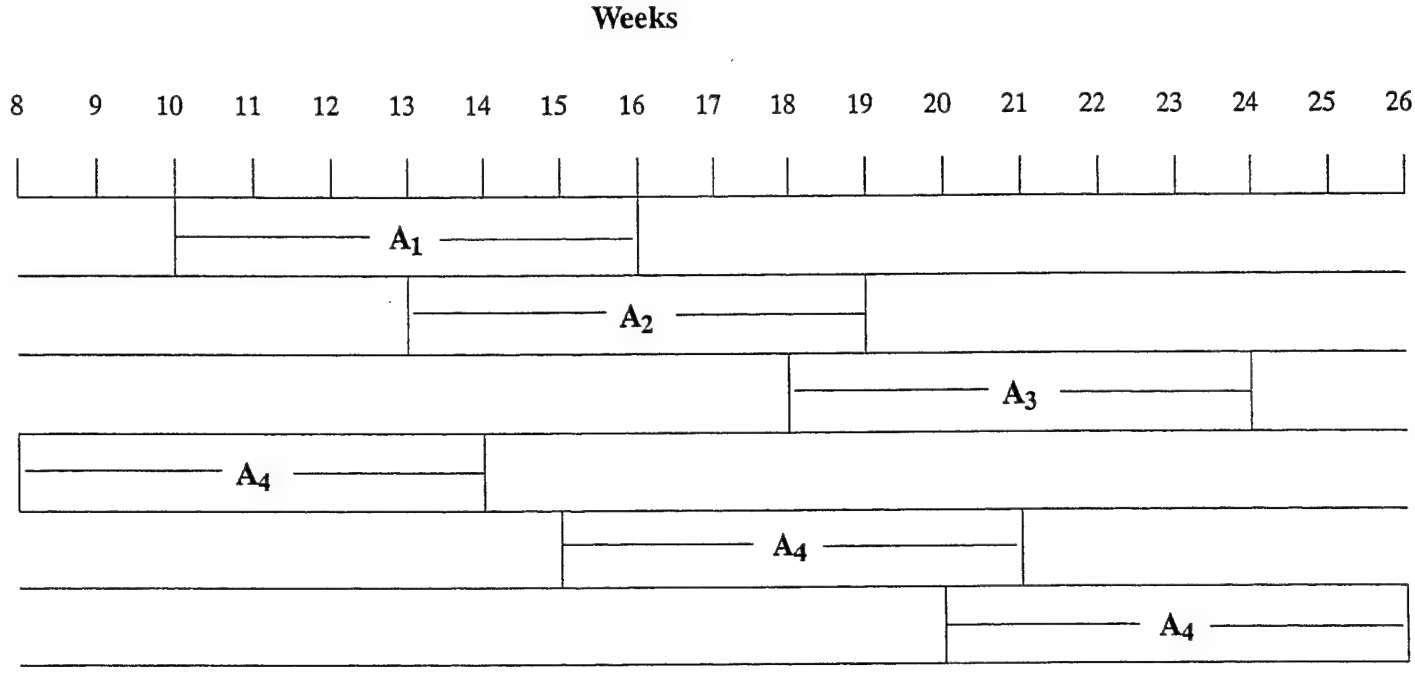


Figure 5. Potential Convening Dates for a Fourth Offering of Course A (Earliest Possible Convening Date is Week 8 and the Offset is 2)

Table 1. The FFG−7 Pipeline Scheduling Problem (Actual Input Data File)

Line	Data	Data Type	No. of Fields	Description of Fields
1	1995 1	1	2	(1) starting fiscal year (2) number of years of the schedule
2	FFG−7	2	1	(1) pipeline name
3	8 4	3	2	(1) number of pipelines (2) number of courses in each pipeline
4	A	4	1	(1) course name (A = HFSYS)
5	2 12	5	2	(1) number of shifts for a given course (2) course length in weeks
6	4 12 0	6	3	(1) number of classes for a given shift (2) offset in weeks between consecutive classes for a given shift (3) carry−over from the previous fiscal year in weeks
7	4 12 0	6	3	same as for line 6
8	B	4	1	(1) course name (B= WSC3)
9	2 12	5	2	same as for line 5
10	5 8 0	6	3	same as for line 6
11	4 8 0	6	3	same as for line 6
12	C	4	1	(1) course name (C= NAVMACS)
13	2 12	5	2	same as for line 5
14	8 4 0	6	3	same as for line 6
15	8 4 0	6	3	same as for line 6
16	D	4	1	(1) course name (D = VRC46)
17	1 2	5	2	same as for line 5
18	8 2 0	6	3	same as for line 6

Table 2. Class Scores for the Schedule Illustrated in Figure 3 (Total Score = 19)

Course–Shift	Class							
	a	b	c	d	e	f	g	h
A1	0	0	0	0				
A2	0	0	0	6				
B1	0	0	0	0	0			
B2	0	0	0	0				
C1	0	0	0	0	0	0	0	3
C2	0	0	0	0	0	0	0	10
D1	0	0	0	0	0	0	0	0

Table 3. Pipeline Scores for the Schedule Illustrated in Figure 3

Pipeline	Target Week	Actual Start Week	Target Violation Score	Gap Score
a	1	1	0	0
b	4	4	0	0
c	7	7	0	0
d	10	11	0	0
e	13	13	0	0
f	16	16	0	0
g	19	19	0	0
h	22	23	0	0

Table 4. Computational Comparison of Pipeline Assistant with the Greedy Algorithm (All times are in seconds on a 50 MHz 486–based PC.)

Problem Name	89_Sup	1_LVL	4_LVL	FFG7	ACT_2	PAS_2
Problem Characteristics						
No. Courses	4	15	4	4	7	10
No. Pipelines	4	4	7	8	5	8
Pipeline Assistant						
final score	0	4	0	37	21	NF[1]
total time	420	1200	1080	420	1440	2820
Greedy Plus Local Improvement						
final score	0	0	0	19	21	278
total time	1	1	1	1	103	210

[1]No feasible solution found.

Table 5. Empirical Analysis of the Algorithms (All times are in seconds on a DecStation 5000/240.)

Algorithms	Problems					
	89_Sup	1_LVL	4_LVL	FFG7	ACT_2	PAS_2
Greedy Plus Local Improvement						
final score	0	0	0	19	21	278
total time	1	1	1	1	10	53
Tabu Search						
initial score	0	2	28	39	44	980
final score	0	2	8	27	26	873
total time	1	131	128	78	120	143
Genetic Algorithm						
initial score	0	2	28	39	44	980
final score	0	2	15	35	35	953
total time	1	64800	21600	46800	162000	122400
Evolutionary Programming						
initial score	0	0	3	28	21	293
final score	0	0	3	28	21	293
total time	18	18	16	16	17	18

Table 6. Empirical Analysis of the Implicit Enumeration Algorithm (All times are in seconds on an Alpha based DecStation.)

Problem Name	Lower Bound	Greedy		Implicit Enumeration		
		Time	Score	Nodes	Time	Score
89_SUP	0	0.01	0	0	0	0
1_LVL	0	0.01	0	0	0	0
4_LVL	0	0.03	0	0	0	0
ACT_2	21	0.02	21	0	0	0
FFG−7	18	0.01	21	2,056,133	1679	18
PAS_2†	220	17	278	3,000,000	3402	278

†terminated after 3,000,000 nodes

Table 7. Characteristics of Test Problems

Problem Name	Number Resources	Number Courses	Number Course Offerings	Course Duration(s)
Test01	9	1	60	54
Test02	9	1	60	54
Test03	9	1	60	54
b10	1	1	20	110
c10	1	6	5 5 5 5 5 5	10 110 15 30 54 80
e05	1	5	5 1 5 2 5	10 15 30 54 80
f01	5	3	5 5 5	30 15 10

Table 8. Comparison of the Simulated Annealing Algorithm and the Greedy Heuristic on the A−School Scheduling Problem

Problem Name	Number of Instructors Needed	
	Simulated Annealing	Greedy Heuristic
Test01	48	41
Test02	48	47
Test03	44	41
b10	10	7
c10	9	7
e05	3	3
f01	No Feasible Schedule	1

Appendix C

Generalized Networks

INTERFACES IN COMPUTER SCIENCE AND OPERATIONS RESEARCH

Advances in Parallel Optimization, Telecommunications and Metaheuristics

EDITED BY

Richard S. BARR,
Jeffery L. KENNINGTON, and
Richard V. HELGASON
Southern Methodist University
Dallas, Texas, USA

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

6

AN EFFICIENT DUAL SIMPLEX OPTIMIZER FOR GENERALIZED NETWORKS

Jeffery L. Kennington and Riad A. Mohamed*

Southern Methodist University
Dallas, Texas 75275-0122

**SABRE Decision Technologies*
DFW Airport, Texas 75261-9616

ABSTRACT

This Chapter describes a specialization of the dual simplex algorithm for the generalized network problem. We use a dual two phase method along with efficient dual partial pricing schemes and specialized routines for the dual ratio test. In comparison with CPLEX 3.0 on a set of ten benchmark problems, we found that our specialized dual code performed 20 times faster than the best CPLEX optimizer. Problems having 10 to 20 thousand arcs are routinely solved in under 10 seconds on a 60MHz DECStation 5000/260 with our rapid dual generalized network optimizer.

1 INTRODUCTION

The mathematical problem of finding a minimal cost flow through a capacitated network is a fundamental problem in both operations research and computer science. A generalization of this model which allows for either gains or losses as flows pass along an arc is known as the *generalized network problem.* Generalized networks have been used to model a wide array of applications in many engineering and economic areas. Applications that involve either gains or losses include electric power carried on transmission lines, cash management that involves the time value of money, and manufacturing processes of varying efficiencies. Other applications allow for flow conversion from one unit to another. Further discussion of a wide variety of applications can be found in Ahuja, Magnanti, and Orlin [1] , Glover, Klingman, and Phillips [11], Glover *et al.* [10], and Mulvey and Zenios [19].

1.1 Problem Description

Let $e_k = (i, j)$ denote an arc from node i to node j having multiplier a_k associated with node i, and b_k associated with node j. If $i = j$, then arc e_k has one end which is incident on node i, with a single multiplier a_k. Let $\mathcal{V} = \{1, 2, \ldots, m\}$ denote a set of m nodes, and let $\mathcal{A} = \{e_1, e_2, \ldots, e_n\}$ denote a set of n arcs. The corresponding generalized network can be represented on a directed graph $< \mathcal{V}, \mathcal{A} >$. For $e_k = (i, j)$ let

$$g_\ell^k = \begin{cases} a_k, & \text{for } \ell = i; \\ b_k, & \text{for } \ell = j; \\ 0, & \text{otherwise.} \end{cases}$$

Let $G = [g^1|g^2|\ldots|g^n]$, be an $m \times n$ matrix called the *node-arc coefficient* matrix. Let the n-component vectors c, l, u, and x denote the arc costs, arc lower bounds ,arc capacities, and arc flows; respectively. Let r be an m-component vector representing the right-hand-side. Let $\mathcal{G} = \{x \in \Re^n : Gx = r, l \leq x \leq u\}$ denote the set of feasible solutions. The generalized network problem can be stated mathematically as $\min_x \{cx : x \in \mathcal{G}\}$.

1.2 Survey of Literature

Generalized networks have the special structure that the basis can be represented graphically as a collection of trees and quasi-trees. In 1973, Glover, Klingman, and Stutz [12], developed the first specialized primal simplex code (NETG), that exploits the special graphical structure using the extended augmented predecessor index procedure. In 1987, Glover *et al.* [10] enhanced NETG, by investigating various rules for the starting strategy, the pivot selection criteria, and degeneracy handling. In 1984, Brown and McBride [6] presented a detailed description of a complete implementation of an efficient primal simplex system specialized for the generalized network problem. The system uses a preorder traversal method in addition to the predecessor, depth, and cycle factor to represent the basis. To enhance the algorithm performance, Brown and McBride included a basis aggregation technique, a dynamic queue scheme for selecting the entering variable, and a big-M starting strategy. In 1985, Engquist and Chang [8] presented a brief description for implementing the primal simplex code GRNET for generalized networks that is based on the labeling procedures of Barr, Glover, and Klingman [3]. Mulvey and Zenios [19] investigated the performance efficiency of the primal simplex generalized network code LPNETG, when using different internal programming tactics such as alternative pivot strategies, column normalization, and the big-M starting

method. In 1988, Nulty and Trick [22] developed the first primal simplex code written in the C language. They use the predecessor, thread, reverse thread, and the level node labels presented in Kennington and Helgason [15] to represent the basis. In 1988, Bertsekas and Tseng [5] proposed a new class of algorithms, that adopt the nonlinear programming relaxation methods, to solve the linear cost generalized network problem. The algorithm, is based on the iterative improvement of the dual cost while maintaining a flow that satisfies complementary slackness. In 1992, Clark *et al.* [7] developed and empirically tested two primal simplex parallel algorithms, GENFLO, and GRNET2, for solving generalized networks. Both codes use a gradient penalty method to find a starting feasible solution.

1.3 Objective of Investigation

Although several papers have been presented that discuss the development of specialized algorithms and software implementations for solving generalized networks, none of these are based on the dual simplex method. The objective of this investigation is to develop and perform an empirical analysis of a specialization of a dual simplex algorithm for the generalized network problem. There are two key issues in the development of a dual procedure: (i) the creation of an effective dual pricing scheme, and (ii) the creation of an efficient scheme for computing an updated row of the coefficient matrix which exploits the underlying network structure.

2 THE DUAL SIMPLEX ALGORITHM

Duality theory in linear programming is well known. Each variable in the primal corresponds to a constraint in the dual, and each constraint in the primal is associated with a dual variable. The dual simplex method has received limited attention, and its role has been limited to sensitivity analysis, parametric programming, and the solution of integer programming problems (see Nemhauser and Wolsey [21] and Parker and Rardin [23]).

There are some instances where the dual method has an advantage over the primal method, see Ali, Padman, and Thiagarajan [2]. The reported results of their dual simplex code, specialized for pure network problems, illustrates improved performance on a subset of the benchmark of standard NETGEN

problems (Klingman, Napier, and Stutz [16]), and an additional set of larger transportation problems.

Under the non-degeneracy assumption, the primal simplex algorithm starts with a basic primal feasible solution and pivots to a new one with an improved objective function. These pivots are repeated until primal optimality is achieved, i.e. obtaining a dual feasible solution. The dual method starts with a dual feasible solution and pivots to a new one with an improved objective function, while attempting to reduce primal infeasibilities. This is repeated until dual optimality (primal feasibility) is achieved. In this section we present the dual simplex algorithm, pricing techniques for selecting the leaving variable, and a specialization for generalized network problems.

Let π_i denote the dual variable associated with the i^{th} conservation of flow constraint, and let λ_j denote the dual variable associated with the lower bound constraint on the j^{th} variable, and let μ_j denote the dual variable associated with the upper bound constraint on the j^{th} variable. Let $\mathcal{D} = \{(\pi \in \Re^m, \lambda \in \Re^n, \mu \in \Re^n) : G^T\pi + \lambda - \mu = c,\ \lambda \geq 0,\ \mu \geq 0\}$, denote the set of dual feasible solutions. Therefore, the dual problem for the generalized network problem is $\max_{\pi,\lambda,\mu} \{r\pi + l\lambda - u\mu :\ (\pi, \lambda, \mu) \in \mathcal{D}\}$. For any $\hat{q} \in \mathcal{V}$, let $q = A(\hat{q})$ denote the index of the associated basic variable x_q, and let $e^{\hat{q}}$ denote the corresponding column of an $m \times m$ identity matrix. Recall that G is an $m \times n$ matrix with full row rank, $g^j \in \Re^m$ denotes the j^{th} column of G, B denotes a basis. Let $\bar{x}$ be an extreme point and let the flow variables be partitioned into *basic* and *non-basic* variables. A non-basic variable must assume one of it's bounds, i.e. $\bar{x}_k = l_k$ or $\bar{x}_k = u_k$. Let $\mathcal{N}_l = \{k : \bar{x}_k = l_u, e_k \in \mathcal{A}\}$, denote the set of indices of the non-basic variables at lower bound, let $\mathcal{N}_u = \{k : \bar{x}_k = u_k, e_k \in \mathcal{A}\}$, denote the set of indices of the non-basic variables at upper bound, and let $\mathcal{N} = \mathcal{N}_l \cup \mathcal{N}_u$. Let $x^N \in \Re^{n-m}$ denote the vector of non-basic variables; let $x^B \in \Re^m$ denote the vector of basic variables; and let $\mathcal{B} = \mathcal{A} \backslash \mathcal{N}$ denote the set of indices of the basic variables. Recall that $\pi \in \Re^m$ denotes the vector of duals associated with the flow conservation constraints $Gx = r$, and that $A(i)$ is the index k, such that e_k is the basic variable associated with constraint i. Let $\sigma_k = c_k - \pi g^k,\ k \in \mathcal{A}$, denote the reduced cost for arc e_k. The dual simplex algorithm can be stated as follows:

Procedure *Dual*
Input : $c, G, l, r, u, \mathcal{B}, \mathcal{N}_l, \mathcal{N}_u$.
Output : $\bar{x}^B, \bar{x}^N, v^*$.
begin
$\bar{\pi} \leftarrow c^B B^{-1};\ \sigma_k \leftarrow c_k - \bar{\pi} g^k,\ k \in \mathcal{N};$
$\bar{r} \leftarrow r - \sum_{k \in \mathcal{N}_u} g^k u_k - \sum_{k \in \mathcal{N}_l} g^k l_k;\ \ \bar{x}^B \leftarrow B^{-1}\bar{r};$

$v^* \leftarrow \sum_{k\in\mathcal{N}_u} c_k u_k + \sum_{k\in\mathcal{N}_l} c_k l_k + \sum_{k\in\mathcal{B}} c_k \bar{x}_k^{\mathcal{B}}$;
$\mathcal{I} \leftarrow \{i \in \mathcal{V} : (\bar{x}_i^{\mathcal{B}} < l_i^{\mathcal{B}}) \text{ or } (\bar{x}_i^{\mathcal{B}} > u_i^{\mathcal{B}})\}$;
while($\mathcal{I} \neq \phi$)**do**
select $\hat{q} \in \mathcal{I}$; $q \leftarrow A(\hat{q})$; / * leaving variable * /
$z \leftarrow e^{\hat{q}} B^{-1}$; / * row $\hat{q}$ of basis inverse * /
$w \leftarrow \text{sign}(u_q - \bar{x}_q) z G$; / * updated row * /
$\mathcal{E} \leftarrow \{j \in \mathcal{N}_l : w_j < 0\} \cup \{j \in \mathcal{N}_u : w_j > 0\}$;
if($\mathcal{E} = \phi$)**then** exit('unbounded dual problem');
$\begin{bmatrix} \delta \\ s \end{bmatrix} \leftarrow \begin{matrix} \max_{j\in\mathcal{E}} \\ \text{argmax}_{j\in\mathcal{E}} \end{matrix} \left\{ \frac{\sigma_j}{w_j} \right\}$; / * ratio test * /
$y \leftarrow B^{-1} g^s$; $\mathcal{P} \leftarrow \{i : y_i \neq 0\}$; / * updated column * /
$\Delta \leftarrow \left\{ \begin{matrix} \frac{(\bar{x}_q - l_q)}{y_{\hat{q}}} & \text{if } \bar{x}_q < l_q \\ \frac{(\bar{x}_q - u_q)}{y_{\hat{q}}} & \text{if } \bar{x}_q > u_q \end{matrix} \right\}$;
$\bar{x}_i^{\mathcal{B}} \leftarrow \bar{x}_i^{\mathcal{B}} - \Delta y_i,\ i \in \mathcal{P};\ \bar{x}_s \leftarrow \bar{x}_s + \Delta$; / * flow update * /
if($\delta < 0$)**then**
$\sigma_k \leftarrow \sigma_k - \delta w_k,\ k \in \mathcal{N};\ \sigma_q \leftarrow \left\{ \begin{matrix} \delta & \text{if } \bar{x}_q = l_q \\ -\delta & \text{if } \bar{x}_q = u_q \end{matrix} \right\}$;
$\bar{\pi} \leftarrow \bar{\pi} + \delta z;\ v^* \leftarrow v^* + \sigma_s \Delta$; / * duals and objective update * /
endif
$\mathcal{B} \leftarrow \mathcal{B} \cup \{s\} \backslash \{q\};\ \mathcal{N}_l \leftarrow \mathcal{N}_l \backslash \{s\};\ \mathcal{N}_u \leftarrow \mathcal{N}_u \backslash \{s\};$ / * basis update * /
if($\bar{x}_q = l_q$)**then** $\mathcal{N}_l \leftarrow \mathcal{N}_l \cup \{q\}$;
if($\bar{x}_q = u_q$)**then** $\mathcal{N}_u \leftarrow \mathcal{N}_u \cup \{q\}$;
$\mathcal{I} \leftarrow \{i \in \mathcal{V} : (\bar{x}_i^{\mathcal{B}} < l_i^{\mathcal{B}}) \text{ or } (\bar{x}_i^{\mathcal{B}} > u_i^{\mathcal{B}})\}$;
endwhile
end.

2.1 A Specialized Implementation for Generalized Networks

The main operations of a dual simplex pivot are selecting the leaving variable, calculating the updated row, determining the entering variable, and updating the flows, the reduced cost, and the inverse of the basis. For generalized networks, most of these operations can be carried out on the tree representing the basis, and the inverse of the basis is never calculated.

Calculating a row of the basis inverse, $z = e^{\hat{q}} B^{-1}$, can be considered as a special case of the dual calculations where the vector c is replaced by $e^{\hat{q}}$ with a single nonzero entry in position $\hat{q}$. For more details on the specialized dual calculations see Mohamed [18]. Let $\mathcal{Z} = \{i : z_i \neq 0\}$, denote the set of indices

for the nonzero entries in z. Let B^q be the basis partition corresponding to the augmented forest component that contains node $\hat{q}$, and let z^q and $e^{\tilde{q}}$, be the matching partitions of z and $e^{\hat{q}}$. Suppose that B^q corresponds to a rooted-tree, T^r. The system $z^q B^q = e^{\tilde{q}}$ can be solved using back substitution, and $\mathcal{Z}$ will only contain the nodes in the subtree component rooted at node $\hat{q}$. Suppose that B^q corresponds to a one-tree, T^o and consider only the cycle of T^o. For notation purpose, assume that the cycle nodes are numbered $1, \ldots, \hat{m}$. First, the z^q values for the cycle nodes are obtained using

$$\begin{aligned} z_1^q &= \frac{1}{\hat{a}_1(1 - w_1 w_2 \cdots w_{\hat{m}})}, \text{ and} \\ z_{i+1}^q &= \frac{-\hat{a}_i}{\hat{b}_i} z_i, \; i = 1, \ldots, \hat{m} - 1. \end{aligned}$$

Then, the z^q values for the remaining nodes in the one-tree are calculated using back substitution. In this case $\mathcal{Z}$ will contain all nodes in the one-tree T^o.

The calculation of the the updated row, $w = zG$, is a dominant operation in the dual pivot. Preliminary profile investigation revealed that, when using standard matrix operations, approximately 50% of the time is spent in calculating w with a complexity of $O(mn)$. This time can be drastically reduced by exploiting the sparsity of z and the special structure of the network problems. Let G^i denote the i^{th} row of G. To exploit the sparsity of z, the updated row calculation is carried out as follows:

$$w = \sum_{i \in \mathcal{Z}} z_i G^i.$$

This reduces the complexity of calculating w to $O(|\mathcal{Z}|n)$, where $|\mathcal{Z}| \ll m$ in most cases. Let $\mathcal{F}^i = \{k : e_k = (i, v) \in \mathcal{A}, v \in \mathcal{V}\}$, the forward star of node i, and let $\mathcal{T}^i = \{k : e_k = (v, i) \in \mathcal{A}, v \in \mathcal{V}\}$, the backward star of node i. Let $\mathcal{W} = \{j : w_j \neq 0\}$. For network problems, G^i can be represented using the forward and the backward stars of node i. This special structure of the network problem has a significant influence on the sparsity of w, that is $\mathcal{W} = \cup_{i \in \mathcal{Z}} (\mathcal{F}^i \cup \mathcal{T}^i)$. This reduces the complexity of calculating the updated row to $O(|\mathcal{Z}|\tilde{n})$, where $\tilde{n}$ is the average number of arcs incident to a node. In practice $\tilde{n} \ll n$.

The entering variable, x_s, and the change in the reduced cost, δ, are determined in the ratio test, which is conducted on the nonbasic variables in $\mathcal{E} \subseteq \mathcal{W}$. The updated column, y, is calculated using the specialized procedure described in Helgason and Kennington [15]. The flow update is carried out only on the basic arcs associated with the nodes in $\mathcal{P}$. For additional information see Mohamed [18]. This is followed by updating the reduced cost of the nonbasic

variables in $\mathcal{W}$, updating the node labels, updating $\mathcal{B}$ and $\mathcal{N}$ to obtain the new basis, and updating the duals.

Let k denote the k^{th} iteration of the dual simplex algorithm. Let $\pi^{(k)}$ be the vector of duals at iteration k, let $B^{(k)}$ be the basis matrix at iteration k, and let $D^{(k)} = [B^{(k)}]^{-1}$. Let $q \in \mathcal{B}$ be the leaving variable, and let $\hat{q}$ denote it's position in B. Therefore, the dual vector at iteration $k+1$ is defined as follows:

$$\pi^{(k+1)} = c^{B^{(k+1)}} D^{(k+1)}, \tag{6.1}$$

where

$$D^{(k+1)} = \left\{ I - \frac{1}{y_p}(y - e^{\hat{q}})e^{\hat{q}^T} \right\} D^{(k)}, \tag{6.2}$$

and

$$c^{B^{(k+1)}} = c^{B^{(k)}} \left\{ I - \left(\frac{c_q - c_s}{c_q} \right) e^{\hat{q}} e^{\hat{q}^T} \right\}. \tag{6.3}$$

Substituting (6.2) and (6.3) into (6.1) we obtain

$$\begin{aligned}
\pi^{(k+1)} &= \left\{ c^{B^{(k)}} - c^{B^{(k)}} \left(\frac{c_q - c_s}{c_q} \right) e^{\hat{q}} e^{\hat{q}^T} \right\} \left\{ D^{(k)} - \frac{1}{y_q}(y - e^{\hat{q}})e^{\hat{q}^T} D^{(k)} \right\} \\
&= c^{B^{(k)}} D^{(k)} - c^{B^{(k)}} \left(\frac{c_q - c_s}{c_q} \right) e^{\hat{q}} e^{\hat{q}^T} D^{(k)} - c^{B^{(k)}} \frac{(y - e^{\hat{q}})}{y_q} e^{\hat{q}^T} D^{(k)} \\
&\quad + c^{B^{(k)}} \left(\frac{c_q - c_s}{c_q} \right) \frac{e^{\hat{q}} e^{\hat{q}^T}}{y_q} (y - e^{\hat{q}})e^{\hat{q}^T} D^{(k)} \\
&= \pi^k + \frac{1}{y_q} \left\{ c^{B^{(k)}} y_q \left(\frac{c_s - c_q}{c_q} \right) e^{\hat{q}} - c^{B^{(k)}} (y - e^{\hat{q}}) \right. \\
&\quad \left. + c^{B^{(k)}} \left(\frac{c_q - c_s}{c_q} \right) e^{\hat{q}} e^{\hat{q}^T} (y - e^{\hat{q}}) \right\} e^{\hat{q}^T} D^{(k)} \\
&= \pi^k + \frac{1}{y_q} \left\{ c^{B^{(k)}} y_q \left(\frac{c_s - c_q}{c_q} \right) e^{\hat{q}} + c^{B^{(k)}} (e^{\hat{q}} - y) \right. \\
&\quad \left. + c^{B^{(k)}} \left(\frac{c_q - c_s}{c_q} \right) e^{\hat{q}} (y_q - 1) \right\} e^{\hat{q}^T} D^{(k)} \\
&= \pi^k + \frac{1}{y_q} \left\{ c^{B^{(k)}} (e^{\hat{q}} - y) - c^{B^{(k)}} e^{\hat{q}} \left(\frac{c_q - c_s}{c_q} \right) \right\} e^{\hat{q}^T} D^{(k)} \\
&= \pi^k + \frac{1}{y_q} \left\{ c_q - c^{B^{(k)}} y - c_q \left(\frac{c_q - c_s}{c_q} \right) \right\} e^{\hat{q}^T} D^{(k)} \\
&= \pi^k + \left(\frac{c_s - c^{B^{(k)}} y}{y_q} \right) e^{\hat{q}^T} D^{(k)}
\end{aligned}$$

$$= \pi^{(k)} + \left(\frac{\sigma_s}{y_q}\right) e^{\hat{q}^T} D^{(k)}$$

Therefore, the dual vector can be updated using $\pi^{(k+1)} = \pi^{(k)} + \left(\frac{\sigma_s}{y_q}\right) z$, and the sparsity of z can be exploited in this update.

2.2 Dual Pricing Strategies

Since all nonbasic variables assume one of their bounds, primal infeasibility occurs only when the basic variables violate the bounds. The dual simplex pricing strategy to select the leaving variable x_q has received very little attention in the literature. Although the selection of the leaving variable appears to be a simple operation, the exact strategy used is very important in the development of an efficient generalized network code.

The standard approach for choosing the leaving variable is to scan the basic variables and determine the variable having maximum bound violation. The index of the leaving variable selected using the *maximum-infeasibility* strategy is

$$q = \text{argmax}_{k \in \mathcal{B}} \left\{ \begin{array}{ll} l_k - \bar{x}_k & \text{if } \bar{x}_k < l_k \\ \bar{x}_k - u_k & \text{if } \bar{x}_k > u_k \end{array} \right\}.$$

Using this approach, every basic variable is scanned at each iteration, which requires a node length search. This section presents techniques which improve this search by performing a partial pricing where only a subset of the basic variables are scanned.

Let $\rho \leq |\mathcal{V}|$ denote the page length. The *fixed-page pricing* technique scans a page of basic variables at a time. If one prices favorable, then it is selected; otherwise, another page is scanned. This continues until either a favorable variable is found or all basic variables have been priced. The list of basic variables are treated in a wrap-around fashion. Let ℓ denote the index of the most recently priced entry, with ℓ initially set to one. The fixed-page pricing technique can be described mathematically as follows:

Procedure : *D_Fixed_Page_Pricing*
Input : ℓ.
Output : ℓ, q.
begin
  $j \leftarrow q \leftarrow \Delta_q \leftarrow 0$;
  while($j \leq |\mathcal{V}|$ & $q = 0$)**do**
    $i \leftarrow 1$; $\hat{\rho} \leftarrow \min\{\rho, |\mathcal{V}|\}$;

while($i \leq \hat{\rho}$)**do** / * price $\hat{\rho}$ basic arcs * /
$k \leftarrow A(\ell)$; / * get basic arc index associated with node ℓ * /

$$\Delta_k \leftarrow \begin{cases} l_k - \bar{x}_k & \text{if } \bar{x}_k < l_k; \\ \bar{x}_k - u_k & \text{if } \bar{x}_k > u_k; \\ 0 & \text{otherwise;} \end{cases}$$ / * get the bound violation value * /

$$\begin{bmatrix} \Delta_q \\ q \end{bmatrix} \leftarrow \underset{\text{argmax}}{\max} \{\Delta_k, \Delta_q\};$$

$i \leftarrow i+1$; $j \leftarrow j+1$; $\ell \leftarrow \text{mod}(\ell, |\mathcal{V}|) + 1$;
endwhile
endwhile
end.

The fixed-page pricing strategy can be enhanced by maintaining multiple pricing candidates. A candidate-queue, which is a cyclic list of varying length, is used to keep the indices of the favorable variables (variables violating their bounds). The *candidate-queue pricing* technique starts with a general scan pricing a set of basic variables to select the most favorable variable (the variable having the maximum bound violation), while placing other favorable variables into the queue. At each iteration a subset of the queue entries are priced. The most favorable variable becomes the leaving variable, and other variables that price favorable are placed back in the queue. A general scan is used to replenish an exhausted queue, where each scan starts where the previous one has ended.

Let Q, called the *candidate-queue*, be a sequence of entries, and let ω denote the maximum allowable size of the candidate-queue. Let $Q[1]$ denote the first entry of the candidate-queue, and let $Q[i \ldots j]$ denote entries i through j. Let $Q\&\&(t)$ denote the concatenation of (t) to the end of Q. Let ρ denote the number of queue candidates to be priced in each iteration. Let ℓ denote the index of the node associated with the most recently priced basic variable, with ℓ initialized to one. The candidate-queue dual pricing technique can be represented as follows:

Procedure : *D_Candidate_Queue_Pricing*
Input : ℓ, Q.
Output : ℓ, q, Q.
begin
$q \leftarrow \Delta_q \leftarrow 0$;
while($Q \neq \phi$ & $q = 0$)**do**
$i \leftarrow 1$; $\hat{\rho} \leftarrow \min\{\rho, |Q|\}$;
while($i \leq \hat{\rho}$)**do** / * price $\hat{\rho}$ queue entries * /
$j \leftarrow Q[1]$; $Q \leftarrow Q[2 \ldots |Q|]$; / * get the first queue entry * /

$k \leftarrow A(j)$; /* get basic arc index associated with node j */

$$\Delta_k \leftarrow \begin{cases} l_k - \bar{x}_k & \text{if } \bar{x}_k < l_k; \\ \bar{x}_k - u_k & \text{if } \bar{x}_k > u_k; \\ 0 & \text{otherwise;} \end{cases}$$ /* get the bound violation value */

$$\begin{bmatrix} q \\ k \end{bmatrix} \leftarrow \begin{matrix} \text{argmax} \\ \text{argmin} \end{matrix} \{\Delta_k, \Delta_q\};$$

if($\Delta_k > 0$)**then** $Q \leftarrow Q$ && (k);

$i \leftarrow i + 1$;

endwhile

endwhile

if($q = 0$)**then** *D_General_scan*(ℓ, q, Q); /* replenish the empty queue */

end.

The mechanism for replenishing an empty queue, is to perform a general pricing scan on κ basic variables, and place the favorable arcs into the queue. The general scan procedure can be stated as follows:

Procedure: *D_General_scan*

Input : ℓ, Q.

Output : ℓ, q, Q.

begin

$q \leftarrow \Delta_q \leftarrow 0$; $j \leftarrow 1$;

while($j \leq |\mathcal{V}|$ & $q = 0$)**do**

$i \leftarrow 1$;

while($i \leq \kappa$ & $|Q| \leq \omega$)**do**

$k \leftarrow A(\ell)$; /* get basic arc index associated with node ℓ */

$$\Delta_k \leftarrow \begin{cases} l_k - \bar{x}_k & \text{if } \bar{x}_k < l_k; \\ \bar{x}_k - u_k & \text{if } \bar{x}_k > u_k; \\ 0 & \text{otherwise;} \end{cases}$$ /* get the bound violation value */

$$\begin{bmatrix} q \\ k \end{bmatrix} \leftarrow \begin{matrix} \text{argmax} \\ \text{argmin} \end{matrix} \{\Delta_k, \Delta_q\};$$

if($\Delta_k > 0$)**then** $Q \leftarrow Q$ && (k);

$i \leftarrow i + 1; j \leftarrow j + 1; \ell \leftarrow \text{mod}(\ell, |\mathcal{V}|) + 1;$

endwhile

endwhile

end.

The above pricing strategies (maximum-infeasibility, fixed-page, candidate-queue) are similar to strategies that have been used in the primal algorithm. For problems having 4000 or more nodes, the pricing strategy can affect the performance of the dual algorithm.

2.3 Steepest Edge Dual Simplex Algorithm

In the simplex method, we proceed from one vertex to an adjacent one along a selected edge. When using standard pricing, the edge is selected such that the objective function improves the most along that edge. In the standard dual simplex method, the size of the bound violation is taken directly to indicate the gradient improvement. This can be regarded as measuring the gradient in the framework of the current basic variables. In the steepest edge method, the selected edge is the one along which the improvement of the objective function is the highest with respect to the original solution space. This involves taking into account the variables that change along the edge. That is, the selected edge is the one that makes the largest, most obtuse angle with the gradient of the objective function.

It is easy to show that the edge directions for a particular basic dual solution are given by the rows of the corresponding basis inverse, B^{-1}. To select edges that are row-length independent, edge normalization was introduced, where weighting factors are used for estimating the edge-length. The steepest-edge pricing is one of a variety of methods for normalized pricing, in which the bound violations are row scaled. However, it is not practical to calculate the edge-lengths at each iteration. Harris [14], used a dynamic subset of the original solution space as a working framework. An approximation of the weighting factors are obtained using the Euclidean norm of the sub-vector consisting of just the components in the current working framework. That is, the Harris' variant of the steepest edge, called *Devex*, chooses the edges that are only approximately steepest. Goldfarb and Reid [13], developed a recurrence formula for the squares of the Euclidean norms of the edge direction when pivoting from one feasible basis to a new one. Forrest and Goldfarb [9], introduced several new steepest edge algorithms for both the primal and the dual simplex methods.

Let $\eta_i = \|e^i B^{-1}\|^2, i = 1, \ldots, m$, denote the length squared of row i of the basis inverse. Following the previously described dual simplex terminology, the steepest edge dual simplex algorithm, using the Goldfarb and Reid recurrence formula, for a given dual feasible solution defined by $\mathcal{B}, \mathcal{N}_l$, and $\mathcal{N}_u$ can be stated as follows:

Procedure : *Dual_Steepest_Edge*
Input : $c, G, l, r, u, \mathcal{B}, \mathcal{N}_l, \mathcal{N}_u$.
Output : $\bar{x}^B, \bar{x}^N, v^*$.
begin
$\bar{\pi} \leftarrow c^B B^{-1}$; $\quad \sigma_k \leftarrow c_k - \bar{\pi} g^k,\ k \in \mathcal{N}$;

$\bar{r} \leftarrow r - \sum_{k \in \mathcal{N}_l} g^k l_k - \sum_{k \in \mathcal{N}_u} g^k u_k$; $\bar{x}^B \leftarrow B^{-1}\bar{r}$;
$v^* \leftarrow \sum_{k \in \mathcal{N}_u} c_k u_k + \sum_{k \in \mathcal{N}_l} c_k l_k + \sum_{k \in \mathcal{B}} c_k \bar{x}_k^B$;
for($i \in \mathcal{B}$)**do** $z \leftarrow e^i B^{-1}$; $\eta_i \leftarrow zz$; /* initiate the row norms */
while($\exists i \in \mathcal{B} : (\bar{x}_i < l_i)$ or $(\bar{x}_i > u_i)$)**do**

$$\hat{q} \leftarrow \text{argmax} \left\{ \begin{array}{ll} \frac{l_{A(k)} - \bar{x}_k^B}{\eta_k} & \text{if } \bar{x}_k^B < l_k \\ \frac{\bar{x}_k^B - u_{A(k)}}{\eta_k} & \text{if } \bar{x}_k^B > u_k \end{array} \right\}, \; k = 1, \ldots, m;$$

$q \leftarrow A(\hat{q})$; $z \leftarrow e^{\hat{q}} B^{-1}$;
$w \leftarrow \text{sign}(u_q - \bar{x}_q) zG$; /* updated row */
$\mathcal{E} \leftarrow \{j \in \mathcal{N}_l : w_j < 0\} \cup \{j \in \mathcal{N}_u : w_j > 0\}$;

$$\left[\begin{array}{c} \delta \\ s \end{array} \right] \leftarrow \begin{array}{c} \max_{j \in \mathcal{E}} \left\{ \frac{\sigma_j}{w_j} \right\} \\ \text{argmax}_{j \in \mathcal{E}} \end{array}; \quad \text{/* Ratio Test */}$$

$y \leftarrow B^{-1} g^s$; $\mathcal{P} = \{i : y_i \neq 0\}$; /* updated column */

$$\Delta \leftarrow \left\{ \begin{array}{ll} \frac{(\bar{x}_q - l_q)}{y_{\hat{q}}} & \text{if } \bar{x}_q < l_q \\ \frac{(\bar{x}_q - u_q)}{y_{\hat{q}}} & \text{if } \bar{x}_q > u_q \end{array} \right\};$$

$\bar{x}_i^B \leftarrow \bar{x}_i^B - \Delta y_i, i \in \mathcal{P}$; $\bar{x}_s \leftarrow \bar{x}_s + \Delta$; /* flow update */

$$\sigma_k \leftarrow \sigma_k - \delta w_k, \; k \in \mathcal{N}; \; \sigma_q \leftarrow \left\{ \begin{array}{ll} \delta & \text{if } \bar{x}_q = l_q \\ -\delta & \text{if } \bar{x}_q = u_q \end{array} \right\};$$

$v^* \leftarrow v^* + \sigma_s \Delta$; /* update the objective value */
$\bar{\pi} \leftarrow \bar{\pi} - \delta z$; $t \leftarrow B^{-1} z$;
for($i = 1, \ldots, m : i \neq \hat{q}$)**do** /* update the norms */

$$\eta_i \leftarrow \eta_i - 2 \left(\frac{y_i}{y_q} \right) t_i + \left(\frac{y_i}{y_q} \right)^2 \eta_{\hat{q}};$$

$$\eta_i \leftarrow \max \left(\eta_i, 1 + \left(\frac{w_i}{w_q} \right)^2 \right);$$

endfor
$\eta_{\hat{q}} \leftarrow \left(\frac{1}{w_q} \right)^2 \eta_{\hat{q}}$;
$\mathcal{B} \leftarrow \mathcal{B} \cup \{s\} \backslash \{q\}$; $\mathcal{N}_l \leftarrow \mathcal{N}_l \backslash \{s\}$; $\mathcal{N}_u \leftarrow \mathcal{N}_u \backslash \{s\}$; /* basis update */
if($\bar{x}_q = l_q$)**then** $\mathcal{N}_l \leftarrow \mathcal{N}_l \cup \{q\}$;
if($\bar{x}_q = u_q$)**then** $\mathcal{N}_u \leftarrow \mathcal{N}_u \cup \{q\}$;
endwhile
end.

In the above dual steepest edge procedure, the standard maximum-infeasibility strategy is used for selecting the leaving variable. The partial pricing techniques were also extended to be used in variants of the steepest edge algorithm.

2.4 An Initial Dual Feasible Solution

A solution $\bar{x}$, defined by $\mathcal{B}, \mathcal{N}_l, \mathcal{N}_u$, is said to be dual feasible if the corresponding dual variables $(\pi, \lambda, \mu) \in \mathcal{D}$. Thus far, we assumed that the dual simplex algorithms are initiated with a dual feasible solution. The strategy needed to obtain a starting dual feasible solution was not evident. A starting basis B for the generalized network problem is obtained by adding an artificial variable $(\zeta_i, i = 1, \ldots, m)$ with zero bounds, to each of the constraints $Gx = r$. Therefore, the set of dual feasible solutions $\mathcal{D}^A = \{\pi \in \Re^m, \lambda^x \in \Re^n, \lambda^\zeta \in \Re^m, \mu^x \in \Re^n, \mu^\zeta \in \Re^m : G^T\pi + \lambda^x - \mu^x = c^x,\ \pi + \lambda^y - \mu^y = c^\zeta,\ \lambda^x \geq 0, \mu^x \geq 0,\ \lambda^y \geq 0, \mu^y \geq 0\}$. In order to assure that $\mathcal{D}^A$ is equivalent to $\mathcal{D}$, the artificial cost vector c^ζ is set to zero. Therefore, for this starting basis, $\pi = 0$, the arc reduced costs are simply the arc costs, and any feasible solution of $\mathcal{D}$, say (π, λ^x, μ^x), has a corresponding feasible solution $(\pi, \lambda^x, \lambda^\zeta, \mu^x, \mu^\zeta)$ where $\lambda^\zeta - \mu^\zeta = \pi$. In the following sections, we introduce techniques for obtaining a dual feasible solution from any starting basis.

Big-M Method

In this approach, the initial dual feasible solution is obtained by setting the original variables $(x_i, i = 1, \ldots, n)$ to be nonbasic at the appropriate bound based on the sign of the corresponding reduced costs. That is, $\mathcal{N}_l = \{j \notin \mathcal{B} : \sigma_j \geq 0\}$, and $\mathcal{N}_u = \{j \notin \mathcal{B} : \sigma_j < 0\}$. Then, the artificial variables $\bar{\zeta} = \bar{r}$, where $\bar{r}$ is the updated right-hand-side. An infinite upper bound is assigned to the original unbounded variables, where infinity is approximated by a large positive constant, usually called big-M. The dual problem becomes $\max_{(\pi,\lambda,\mu)\in\mathcal{D}^A} \{r\pi + l\lambda - u\mu\}$, and optimality is achieved when all variables satisfy primal feasibility. Since the artificial variables have zero bounds, the corresponding flows will be forced to zero at optimality. Also, variables with big-M upper bounds will be driven to a smaller value. Other big-M approaches have been suggested to obtain a dual feasible starting basis (Murty [20]).

Two Phase Method

The big-M approach for setting the initial basis is not recommended because of difficulties in making the appropriate choice of M. To avoid this critical setting of the big-M value, a two phase approach is usually used. In phase 1 a new problem associated with the original dual problem is defined whose optimal solution is dual feasible for the original problem. The initial solution is obtained by setting the original variables $(x_i, i = 1, \ldots, n)$ to be nonbasic at

the appropriate bound based on the sign of the corresponding reduced costs. Unbounded variables with negative reduced costs are assigned to their lower bounds, resulting in a dual infeasibility equal to the sum of the corresponding reduced cost. Therefore, $\mathcal{N}_u = \{j \notin \mathcal{B} : \sigma_j < 0, u_k \text{ is finite}\}$, $\mathcal{I} = \{j \notin \mathcal{B} : \sigma_j < 0, u_k \text{ is undefined}\}$, and $\mathcal{N}_l = \{j \notin \mathcal{B} : \sigma_j \geq 0\} \cup \mathcal{I}$. The phase 1 objective function is obtained by temporarily setting the lower and upper bounds to zero for all variables and introducing a new right-hand-side, $\bar{r} = \sum_{k \in \mathcal{I}} \{-g^k\}$. Therefore, the artificial variables $\bar{\zeta} = \bar{r}$, and the phase 1 objective is to $\max_{(\pi,\lambda,\mu) \in \mathcal{D}^A} \{\bar{r}\pi\}$. This approach is similar to and influenced by the two phase method developed by Professor Bixby in CPLEX 3.0. Let σ_j denote the current reduced cost for $j \in \mathcal{I}$. The contribution to dual infeasibility by x_j is $(0 - (\sigma_j - \delta w_j))$. Thus, the total sum of infeasibility is $\sum_{j \in \mathcal{I}} (\delta w_j - \sigma_j)$. Differentiating with respect to δ, the rate of change of the sum of dual infeasibilities is $\sum_{j \in \mathcal{I}} \{w_j\} = \sum_{j \in \mathcal{I}} \{-z g^j\} = z \sum_{j \in \mathcal{I}} \{-g^j\}$. Let x_q be the leaving variable. Therefore, the rate of change of the objective function is $x_q = e^{\hat{q}} \bar{x} = e^{\hat{q}} B^{-1} \bar{r} = z(r - \sum_{j \in \mathcal{N}_u} g^j u_j - \sum_{j \in \mathcal{N}_l} g^j l_j)$. Setting the upper and lower bounds on all variables to zero, and setting the right-hand-side to be defined in terms of the variables that violate dual feasibility, results in having the rate of change of the objective function to be the same as the rate of change of the sum of dual infeasibilities. This approach can be applied to any basis, and it is not restricted to an all artificial starting basis. Another intuitive interpretation of this setting is that for any $j \in \mathcal{I}$ the reduced cost $\sigma_j = c_j - \pi g^j < 0$. The objective function to drive σ_j to be nonnegative is $\min \{\pi g^j\}$. Therefore, for all the dual infeasible variables the objective function is $\min\{\pi \sum_{j \in \mathcal{I}} g^j\}$.

The computational steps of the dual simplex algorithm in phase 1 are modified to incorporate dual infeasibilities. The pivoting rules are as follows:

1. the leaving variable assumes the closest of the upper or lower bound, and the reduced cost will assume the correct sign. Unbounded variables are allowed to move only to the lower bound.
2. the basic variables remain dual feasible after updating the basis.
3. the entering variable assumes a dual feasible reduced cost.

In this approach the total number of infeasibilities is reduced without restricting the total amount of infeasibility. Other methods that reduce the total amount of infeasibility require fewer iterations, but need some extra computational effort. Belling-Seib [4] presents similar ideas for the primal simplex method.

Following the previous terminology, the phase 1 dual simplex algorithm can be stated as follows:

Procedure *Dual_Phase 1*
Input : $c, G, l, r, u, \mathcal{B}, \mathcal{N}$.
Output : $\bar{x}^B, \bar{x}^N, v^*$.
begin
$\bar{\pi} \leftarrow c^B B^{-1}$; $\sigma_k \leftarrow c_k - \bar{\pi} g^k,\ k \in \mathcal{N}$;
$\mathcal{N}_u \leftarrow \{k \in \mathcal{N} : \sigma_k < 0, u_k \neq \infty\}$; $\mathcal{I} \leftarrow \{k \in \mathcal{N} : \sigma_k < 0, u_k = \infty\}$;
$\mathcal{N}_l \leftarrow \{k \in \mathcal{N} : \sigma_k \geq 0\} \cup \mathcal{I}$;
$\tilde{r} \leftarrow \sum_{k \in \mathcal{I}} \{-g^k\}$; $\bar{x} \leftarrow B^{-1}\tilde{r}$; $v^* \leftarrow \sum_{k \in \mathcal{I}} \sigma_k$;
while($\mathcal{I} \neq \phi$)**do**
$\hat{q} \leftarrow \text{argmax}_{k \in \mathcal{V}} \left\{ \begin{array}{ll} -\bar{x}_k^{\mathcal{B}} & \text{if } \bar{x}_k^{\mathcal{B}} < 0 \\ \bar{x}_k^{\mathcal{B}} & \text{if } \bar{x}_k^{\mathcal{B}} > 0, u_k^{\mathcal{B}} \neq \infty \end{array} \right\}$; /* leaving variable */
$q \leftarrow A(\hat{q})$; $z \leftarrow e^{\hat{q}} B^{-1}$; /* row $\hat{q}$ of basis inverse */
$w \leftarrow \text{sign}(-\bar{x}_q) z G$; /* updated row */
$\mathcal{E} \leftarrow \{j \in \mathcal{N}_l : w_j < 0\ ,\ \sigma_j \geq 0\} \cup \{j \in \mathcal{N}_u : w_j > 0\ ,\ \sigma_j < 0\}$;
if($\mathcal{E} = \phi$)**then** $\mathcal{E} \leftarrow \{j \in \mathcal{I} : w_j > 0\}$;
$\left[\begin{array}{c} \delta \\ s \end{array} \right] \leftarrow \begin{array}{c} \max_{j \in \mathcal{E}} \\ \text{argmax}_{j \in \mathcal{E}} \end{array} \left\{ \frac{\sigma_j}{w_j} \right\}$; /* ratio test */
$y \leftarrow B^{-1} g^s$; $\mathcal{P} \leftarrow \{i : y_i \neq 0\}$; /* updated column */
if($\bar{x}_q < 0$)**then** $\mathcal{N}_l \leftarrow \mathcal{N}_l \cup \{q\}$;
if($\bar{x}_q > 0$)**then** $\mathcal{N}_u \leftarrow \mathcal{N}_u \cup \{q\}$;
$\Delta \leftarrow \frac{\bar{x}_q}{y_{\hat{q}}}$; $\bar{x}_i^B \leftarrow \bar{x}_i^B - \Delta y_i,\ i \in \mathcal{P}$; $\bar{x}_s \leftarrow \Delta$; /* flow update */
if($\delta > 0$)**then** $\sigma_k \leftarrow \sigma_k - \delta w_k,\ k \in \mathcal{N}$; $\sigma_q \leftarrow \text{sign}(-\bar{x}_q)\delta$;
$\mathcal{B} \leftarrow \mathcal{B} \cup \{s\} \backslash \{q\}$; $\mathcal{N}_l \leftarrow \mathcal{N}_l \backslash \{s\}$; $\mathcal{N}_u \leftarrow \mathcal{N}_u \backslash \{s\}$; /* basis update */
for($k \in \mathcal{I}$ & $\sigma_k \geq 0$)**do**
$\bar{x}^B \leftarrow \bar{x}^B - B^{-1} g^k$; /* adjust the flow */
$\mathcal{I} \leftarrow \mathcal{I} \backslash \{k\}$; /* variable k became feasible */
endfor
endwhile
dual$(c, G, l, r, u, \mathcal{B}, \mathcal{N}, \bar{x}^B, \bar{x}^N, v^*)$;
end.

The search for the leaving variable in phase 1 of the dual simplex method is enhanced by exploiting the infrequency (small number) of the dual infeasibilities and the basis structure of generalized networks. The search is limited to small parts of the basis components associated with the nodes in $\hat{\mathcal{I}} = \{i \in \mathcal{V} : g_i^j \neq 0, j \in \mathcal{I}\}$; precisely the path from a node in $\hat{\mathcal{I}}$ to the root of the corresponding rooted-tree component, and the nodes in the cycle along with the nodes in the path from a node in $\hat{\mathcal{I}}$ to the cycle of the corresponding one-tree component.

3 BENCHMARK PROBLEMS AND TESTING ENVIRONMENT

A set of generalized network test problems were randomly generated using a modified version of NETGEN by Klingman *et al.* [16]. The modified code, generates a variety of generalized network problems based on the user controlled problem description. The main inputs are the number of nodes, the number of arcs, the number of source nodes, the number of sink nodes, the number of transshipment sources, and the number of transshipment sinks. In addition, the minimum supply and the maximum supply specifications bounds the total supply. The % arcs with high cost parameter specifies the number of arcs having the maximum cost, and the remaining arc costs are randomly generated on a user specified interval. Similarly, the % arcs with bounds parameter specifies the percentage of bounded arcs, where the generated arc bounds are uniformly distributed on a given interval. A seed is supplied to a random number generator that allows one to reproduce the problems.

Our modified code can be divided into two main segments. In the first segment, a skeleton network that guarantees the problem connectivity and feasibility is generated. Initially, the total supply is randomly distributed among the source nodes, including the transshipment source nodes. Then, for each source node a chain is created to carry the flow through a series of distinct transshipment nodes, to be randomly allocated among a randomly chosen subset of sink nodes. These chains are mutually exclusive except for the sink nodes. In the second segment, randomly generated arcs are appended to the network to fill out the total required arcs. The final network will contain approximately the required number of arcs. For more detail about generating the arc costs and bounds see Klingman *et al.* [16]. For the skeleton arcs, the from and the to multipliers are set to 1 and -1, respectively. For each of the remaining arcs, the two multipliers are randomly selected from the continuous interval $[-1, 1)$, while insuring that the first multiplier is nonzero.

The comparison and examination of the efficiency for different solution methods is of practical importance, and the selection of the test set used in the comparison is very crucial. Therefore, we generated ten test problems of various sizes and structures. The number of nodes ranged from 1,000 to 15,000, and the required number of arcs ranged from 3,000 to 90,000. The characteristics of the test problems are given in Tables 1 and 2. The naming convention is $G\_\hat{m}\_\hat{n}$ for a generalized problem having $\hat{m}$K nodes, and $\hat{n}$K arcs. All testing was performed on a 60 MHz DECStation 5000/260 running the Ultrix 4.3a operating system. The reported performance measures are the number of pivots

Table 1 Characteristics of test problems one through five.

Specification	Problem				
	G_01_03	G_01_06	G_02_06	G_02_12	G_04_12
Seed	85	173	319	51	920
Nodes	1K	1K	2K	2K	4K
Arcs	3K	6K	6K	12K	12K
Sources	100	100	200	200	200
Sinks	100	100	200	200	200
Trans. Sources	50	50	100	100	100
Trans. Sinks	50	50	100	100	100
Chain Length	8	8	10	20	25
Min Cost	-1K	-2K	-1K	-3K	-4K
Max Cost	10K	10K	10K	10K	10K
Min Supply	10K	10K	50K	50K	50K
Max Supply	20K	20K	50K	50K	70K
Min Arc Bounds	10	10	10	10	10
Max Arc Bounds	1K	1K	1K	1K	1K
High Cost Arcs	58%	40%	29%	95%	78%
Bounded Arcs	70%	99%	87%	73%	92%

Table 2 Characteristics of test problems six through ten.

Specification	Problem				
	G_04_20	G_09_20	G_09_45	G_15_45	G_15_90
Seed	11	194	239	357	421
Nodes	4K	9K	9K	15K	15K
Arcs	20K	20K	45K	45K	90K
Sources	200	200	500	2000	2000
Sinks	200	200	500	2000	2000
Trans. Sources	100	100	250	1000	1000
Trans. Sinks	100	100	250	1000	1000
Chain Length	25	50	100	200	400
Min Cost	-2K	-1K	-3K	-3K	-3K
Max Cost	10K	30K	30K	30K	30K
Min Supply	70K	80K	90K	90K	90K
Max Supply	80K	90K	95K	95K	99K
Min Arc Bounds	10	10	100	100	100
Max Arc Bounds	1K	1K	1K	2K	3K
High Cost Arcs	73%	20%	84%	75%	80%
Bounded Arcs	60%	30%	30%	60%	80%

(or iterations) and the user central processing time rounded to two decimal places. The user cpu time is obtained using the standard Ultrix function *getrusage*, that returns information describing the system resource utilization. This information includes the cumulative user cpu time, measured in microseconds, consumed to the point of the function call. In the implementation of the simplex algorithm, getrusage is called immediately preceding and immediately following the call to the optimizer. The user cpu time consumed is obtained by subtracting the first call returned time from the second call returned time.

4 COMPUTATIONAL EXPERIENCE WITH CPLEX 3.0

To provide a comparison base, the test problems were solved using the four optimization algorithms available in CPLEX 3.0 (a state-of-the-art solver [24].) This version includes efficient implementations of the primal simplex, the dual simplex, the network simplex, and the barrier algorithms. All CPLEX runs were made using the default parameter settings, which activate the presolver and the aggregator problem preprocessors. The presolver helps simplify, reduce, and eliminate redundancies in the problem presentation. The aggregator attempts to make simplifying substitutions that reduce the basis size.

In the CPLEX dual simplex optimizer, the dual problem is solved using the primal simplex presentation, where all the linear algebra is carried out on the associated primal basis. The network optimizer, recognizes and exploits the embedded network structure of the problem, and uses an efficient specialized algorithm on the pure network portion to obtain an advanced start for the dual optimizer. The barrier optimizer, is an efficient implementation of the primal-dual method fully integrated with the predictor corrector basis crossover of Megiddo [17].

The empirical results for solving the test problems using the four CPLEX solvers are presented in Table 3, where the numbers in parenthesis represent the number of phase I pivots. From the first four problems, it is clear that the barrier solver performs poorly for generalized network problems, and it was not run for the remaining ones. The CPLEX dual simplex solver is approximately two times faster than the primal simplex solver for all problems except G_01_03, G_15_45 and G_15_90. The over all time performance is 1.4 times faster than the primal algorithm. Using the default settings, the CPLEX network solver

failed to extract any network component for three of the test problems, and it terminated with an unbounded network for problems G_09_45 and G_15_45.

5 EMPIRICAL ANALYSIS

The specialized big-M dual simplex algorithm is implemented in C, and the performance in solving the ten test problems, using the three pricing techniques, is shown in Tables 4, 5, and 6. For all testing M is set to 10^9. Comparing the big-M standard (non-normalized) dual RAMSES, using the three pricing strategies, and the CPLEX 3.0 dual solver, it is clear that the dual RAMSES with a candidate-queue pricing is the superior. For the G_15_90, it performed approximately 30 times faster than CPLEX, 13 times faster than RAMSES with a maximum-infeasibility pricing strategy, and 28% faster than RAMSES with a fixed-page pricing strategy. On average, for the ten problems it performed 20 times faster than CPLEX, 9 times faster than RAMSES with the maximum-infeasibility pricing strategy, and 25% faster than RAMSES with fixed-page pricing strategy. The fixed-page pricing strategy resulted in only a 2% increase in the total number of pivots, while the candidate-queue pricing strategy resulted in a 19% increase in the total number of pivots over the maximum-infeasibility pricing strategy.

For the three pricing strategies, the standard (non-normalized) RAMSES performed consistently much faster, and required fewer pivots than the steepest edge RAMSES. This can be ascribed to the generated problem characteristics of having arc multipliers in the range [-1,1], and uniformly distributing the arcs among the network nodes, which results in basis inverse rows having approximately the same length.

The specialized two phase dual simplex algorithm is implemented in C and the performance in solving the ten test problems, using the three pricing techniques, is shown in Tables 7, 8, and 9. It is clear that the two phase dual RAMSES with a candidate-queue pricing strategy is the superior implementation. For the G_15_90, it performed approximately 31 times faster than CPLEX, 13 times faster than RAMSES with a maximum-infeasibility pricing strategy, and 29% faster than RAMSES with a fixed-page pricing strategy. On average, for all test problems it performed 22 times faster than CPLEX, 10 times faster than RAMSES with a maximum-infeasibility pricing strategy, and 22% faster than RAMSES with a fixed-page pricing strategy.

Table 3 Solving generalized networks using CPLEX version 3.0

Prob. Name	Primal		Dual		Network		Barrier		Objective Value
	Pivots	Time	Pivots	Time	Pivots	Time	Iter	Time	
G_01_03	975(295)	2.90	647(34)	2.73	647	2.53	20	45	1.0676511e08
G_01_06	2162(311)	7.70	1088(13)	4.67	N/A[a]	N/A	16	151	3.6451339e07
G_02_06	2408(765)	11.70	1526(43)	8.85	1527	8.78	16	373	5.9484944e08
G_02_12	3399(608)	21.95	1294(34)	11.68	N/A[a]	N/A	19	1955	7.8177439e08
G_04_12	4968(1678)	79.15	2748(68)	33.10	2744	37.82	-	-	1.3194873e09
G_04_20	5692(1213)	60.20	2124(243)	29.52	2125	32.52	-	-	1.1445576e09
G_09_20	6394(2538)	176.53	3212(94)	76.53	3194	76.08	-	-	5.2429920e09
G_09_45	7694(2205)	249.03	3958(186)	107.97	238[b]	8.63	-	-	1.4578699e10
G_15_45	16061(5941)	999.37	13106(236)	786.02	1797[b]	10.95	-	-	2.2251216e11
G_15_90	22636(8101)	1576.32	15055(94)	1167.02	15107	1277.27	-	-	2.2564331e11
Totals	72389(23655)	3184.85	44758(1045)	2226.09	-	-	-	-	-

[a]Small or non-existent network
[b]Unbounded network

Table 4 Results for big-M dual RAMSES with maximum-infeasibility pricing.

Prob. Name	CPLEX (DUAL)		Standard			Time Ratio	Steepest Edge			Time Ratio
	Pivots	Time	Pivots		Time		Pivots		Time	
	Total	[1]	Total	Deg.	[2]	[1]/[2]	Total	Deg.	[3]	[1]/[3]
G_01_03	647(34)	2.73	830	107	0.69	4.0	971	122	0.98	2.8
G_01_06	1088(13)	4.67	1340	85	1.29	3.6	1381	89	1.68	2.8
G_02_06	1526(43)	8.85	1977	195	3.35	2.6	2266	230	4.82	1.8
G_02_12	1294(34)	11.68	1660	465	3.98	2.9	1901	571	4.72	2.5
G_04_12	2748(68)	33.10	3389	377	11.49	2.9	3867	418	16.59	2.0
G_04_20	2124(243)	29.52	3145	677	12.14	2.4	3247	645	12.78	2.3
G_09_20	3212(94)	76.53	4900	386	47.87	1.6	5388	317	61.66	1.2
G_09_45	3958(186)	107.97	6195	2244	63.88	1.7	6725	2315	75.52	1.4
G_15_45	13106(236)	786.02	20978	6788	371.47	2.1	25083	7724	530.99	1.5
G_15_90	15055(94)	1167.02	25566	13230	492.77	2.4	25540	11229	559.72	2.1
Totals	44740(1045)	2226.09	69980	24554	1008.93	2.2	76369	23660	1269.46	1.8

Table 5 Results for big-M dual RAMSES with fixed-page pricing.

Prob. Name	CPLEX (DUAL)		Standard			Time Ratio	Steepest Edge			Time Ratio
	Pivots	Time	Pivots		Time		Pivots		Time	
	Total	[1]	Total	Deg.	[2]	[1]/[2]	Total	Deg.	[3]	[1]/[3]
G_01_03	647(34)	2.73	948	114	0.43	6.3	1001	115	0.62	4.4
G_01_06	1088(13)	4.67	1444	86	0.80	5.8	1444	89	1.02	4.6
G_02_06	1526(43)	8.85	2502	197	1.50	5.9	2509	206	1.87	4.7
G_02_12	1294(34)	11.68	2185	545	2.61	4.5	2083	563	2.43	4.8
G_04_12	2748(68)	33.10	4500	416	4.95	6.7	4624	380	8.99	3.7
G_04_20	2124(243)	29.52	4108	755	4.88	6.0	4097	722	6.98	4.2
G_09_20	3212(94)	76.53	6693	408	13.77	5.6	6449	392	21.70	3.5
G_09_45	3958(186)	107.97	7684	2509	15.43	7.0	7687	2498	23.94	4.5
G_15_45	13106(236)	786.02	23198	7017	42.66	18.4	24640	7425	56.54	13.9
G_15_90	15055(94)	1167.02	25738	11187	49.74	23.5	26034	11243	58.35	20.0
Totals	44740(1045)	2226.09	79000	23234	136.77	16.3	80568	23633	182.44	12.3

Table 6 Results for big-M dual RAMSES with candidate-queue pricing.

Prob. Name	CPLEX (DUAL)		Standard			Time Ratio	Steepest Edge			Time Ratio
	Pivots	Time	Pivots		Time	[1]/[2]	Pivots		Time	[1]/[3]
	Total	[1]	Total	Deg.	[2]		Total	Deg.	[3]	
G_01_03	647(34)	2.73	981	121	0.44	6.2	1042	123	0.64	4.3
G_01_06	1088(13)	4.67	1620	84	1.04	4.5	1600	96	1.27	3.7
G_02_06	1526(43)	8.85	2474	236	1.17	7.6	2599	237	1.75	5.1
G_02_12	1294(34)	11.68	2249	600	1.91	6.1	2371	660	2.53	4.6
G_04_12	2748(68)	33.10	4561	411	3.45	9.6	4637	410	5.99	5.5
G_04_20	2124(243)	29.52	3811	742	3.35	8.8	3887	735	4.30	6.9
G_09_20	3212(94)	76.53	6407	421	11.26	6.8	6467	413	28.73	2.7
G_09_45	3958(186)	107.97	7644	2613	11.00	9.8	7753	2706	16.56	6.5
G_15_45	13106(236)	786.02	26115	8149	33.80	23.3	26271	8154	51.99	15.1
G_15_90	15055(94)	1167.02	27282	10992	38.97	30.0	27832	11206	55.88	20.9
Totals	44740(1045)	2226.09	83144	24369	106.39	20.9	84459	24740	169.64	13.1

Table 7 Results for two phase dual RAMSES with maximum-infeasibility pricing.

Prob. Name	CPLEX (DUAL)		Standard			[1]	Steepest Edge			[1]
	Pivots	Time	Pivots		Time	—	Pivots		Time	—
	Total	[1]	Total	Deg.	[2]	[2]	Total	Deg.	[3]	[3]
G_01_03	647(34)	2.73	813(39)	103	0.68	4.0	938(40)	115	0.93	2.9
G_01_06	1088(13)	4.67	1337(13)	85	1.23	3.8	1376(12)	87	1.76	2.7
G_02_06	1526(43)	8.85	1948(50)	193	3.31	2.7	2224(51)	226	4.26	2.1
G_02_12	1294(34)	11.68	1662(36)	446	4.00	2.9	1923(36)	579	4.51	2.6
G_04_12	2748(68)	33.10	3410(72)	387	12.56	2.6	3905(72)	406	18.30	1.8
G_04_20	2124(243)	29.52	3012(288)	558	10.51	2.8	3206(288)	530	14.85	2.0
G_09_20	3212(94)	76.53	4869(173)	361	40.69	1.9	5356(173)	297	62.08	1.2
G_09_45	3958(186)	107.97	5939(242)	2049	62.26	1.7	6565(245)	2122	76.23	1.4
G_15_45	13106(236)	786.02	20572(309)	6657	371.17	2.1	24088(312)	7330	537.78	1.5
G_15_90	15055(94)	1167.02	24648(119)	12469	496.94	2.3	24552(119)	10500	533.64	2.2
Totals	44740(1045)	2226.09	68210(1341)	23308	1003.35	2.2	74133(1348)	22192	1254.34	1.8

Table 8 Results for two phase dual RAMSES with fixed-page pricing.

Prob. Name	CPLEX (DUAL)		Standard			[1]	Steepest Edge			[1]
	Pivots	Time	Pivots		Time	—	Pivots		Time	—
	Total	[1]	Total	Deg.	[2]	[2]	Total	Deg.	[3]	[3]
G_01_03	647(34)	2.73	975(39)	110	0.42	6.5	1017(40)	117	0.58	4.7
G_01_06	1088(13)	4.67	1445(13)	88	0.84	5.6	1446(13)	91	1.01	4.6
G_02_06	1526(43)	8.85	2502(50)	198	1.50	5.9	2548(51)	204	1.93	4.6
G_02_12	1294(34)	11.68	2252(36)	546	2.50	4.7	1988(36)	521	2.70	4.3
G_04_12	2748(68)	33.10	4368(72)	415	4.59	7.2	4496(72)	369	9.69	3.4
G_04_20	2124(243)	29.52	4072(288)	762	5.23	5.6	3949(283)	605	8.11	3.6
G_09_20	3212(94)	76.53	6646(173)	383	13.11	5.8	6963(173)	401	30.94	2.5
G_09_45	3958(186)	107.97	7506(242)	2387	16.32	6.6	7466(245)	2357	24.48	4.4
G_15_45	13106(236)	786.02	22778(312)	8014	40.37	19.5	25099(312)	7553	71.05	11.1
G_15_90	15055(94)	1167.02	25258(119)	10929	49.24	23.7	26027(119)	10962	63.53	18.4
Totals	44740(1045)	2226.09	77802(1344)	23832	134.12	16.6	80999(1344)	23180	214.02	10.4

Table 9 Results for two phase dual RAMSES with candidate-queue pricing.

Prob. Name	CPLEX (DUAL)		Standard			[1]	Steepest Edge			[1]
	Pivots	Time	Pivots		Time	—	Pivots		Time	—
	Total	[1]	Total	Deg.	[2]	[2]	Total	Deg.	[3]	[3]
G_01_03	647(34)	2.73	956(39)	120	0.44	6.2	1036(39)	123	0.66	4.1
G_01_06	1088(13)	4.67	1554(13)	85	1.05	4.4	1577(13)	92	1.29	3.6
G_02_06	1526(43)	8.85	2461(50)	237	1.13	7.8	2519(50)	238	1.61	5.5
G_02_12	1294(34)	11.68	2216(36)	628	1.88	6.2	2337(36)	664	2.58	4.5
G_04_12	2748(68)	33.10	4571(72)	419	3.59	9.2	4799(72)	423	7.57	4.4
G_04_20	2124(243)	29.52	3591(288)	613	3.01	9.8	3612(283)	606	4.28	6.9
G_09_20	3212(94)	76.53	6387(173)	401	10.90	7.0	6450(173)	387	27.24	2.8
G_09_45	3958(186)	107.97	7447(242)	2467	10.48	10.3	7465(245)	2474	16.14	6.7
G_15_45	13106(236)	786.02	25826(312)	8014	33.29	23.6	26099(312)	8114	58.94	13.3
G_15_90	15055(94)	1167.02	26846(119)	10786	38.09	30.6	27283(119)	10873	59.96	19.5
Totals	44740(1045)	2226.09	81855(1344)	23770	103.86	21.5	83177(1342)	23994	180.27	12.4

6 SUMMARY AND CONCLUSIONS

Generalized networks are used to model a diversity of practical problems that include class scheduling, machine loading, manpower planning, resource distribution systems, and currency exchange. This paper describes the development of efficient techniques for solving generalized network problems. As in the primal simplex algorithm, the dual simplex method is initiated by an all artificial basis. In Section 2.4 we introduce two approaches to obtain a dual feasible solution from any starting basis. In the two approaches, the nonbasic variables are assigned to the appropriate bound based on the the sign of the corresponding reduced costs. In the first approach, a big-M value is used to approximate the upper bound for unbounded variables. For the three dual pricing strategies, the big-M dual RAMSES code with a candidate-queue is superior. It achieves a performance 30 times faster than the CPLEX 3.0 dual optimizer for problem G_15_90, and 20 times faster for all the benchmark problems. However, the success of the big-M method is contingent upon the selection of a proper big-M value. In the second approach, this critical selection is avoided by using a two phase method, where dual infeasibilities are temporarily permitted in phase 1 for unbounded variables with negative reduced cost. For the three dual pricing strategies, the two phase dual algorithm with a candidate-queue is superior. It achieves a performance 31 times faster than the CPLEX dual optimizer for problem G_15_90, and 22 times faster for the ten test problems.

Acknowledgements

This work was partially supported by the Office of Naval Research under Grant Number N00014-95-1-0645.

REFERENCES

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ., 1993.

[2] A. Ali, R. Padman, and H. Thiagarajan. Dual Algorithms for Pure Network Problems. *Operations Research*, 37(1):159–179, 1989.

[3] R. Barr, F. Glover, and D. Klingman. Enhancement of Spanning Tree Labeling Procedures for Network Optimization. *INFORS*, 17:66–85, 1979.

[4] K. Belling-Seib. An Improved General Phase-1 Method in Linear Programming. *European Journal of Operational Research*, 36:101–106, 1988.

[5] D. Bertsekas and P. Tseng. Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems. *Operations Research*, 36(1):93–114, 1988.

[6] G. Brown and R. McBride. Solving Generalized Networks. *Management Science*, 30(12):1497–1523, 1984.

[7] R. Clark, J. Kennington, R. Meyer, and M. Ramamurti. Generalized Networks: Parallel Algorithms and an Empirical Analysis. *ORSA Journal on Computing*, 4(2):132–145, 1992.

[8] M. Engquist and M. Chang. New Labeling Procedures for The Basis Graph in Generalized Networks. *Operations Research Letters*, 4(4):151–155, 1985.

[9] J. Forrest and D. Goldfarb. Steepest-edge Simplex Algorithm for Linear Programming. *Mathematical Programming*, 57:341–374, 1992.

[10] F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized Networks: A Fundamental Computer-Based Planning Tool. *Management Science*, 24(12):1209–1220, 1978.

[11] F. Glover, D. Klingman, and N. Phillips. *Network Models in Optimization and Their Applications in Practice.* John Wiley & Sons, New York, 1992.

[12] F. Glover, D. Klingman, and J. Stutz. Extensions of the Augmented Predecessor Index Method to Generalized Network Problems. *Transportation Science*, 7(4):377–384, 1973.

[13] D. Goldfarb and J. Reid. A Practicable Steepest-Edge Simplex Algorithm. *Mathematical Programming*, 12:361–371, 1977.

[14] P. Harris. Pivot Selection Methods of the DEVEX LP Code. *Mathematical Programming*, 5:1–28, 1973.

[15] J. Kennington and R. Helgason. *Algorithms for Network Programming.* John Wiley & Sons, New York, 1980.

[16] D. Klingman, A. Napier, and J. Stutz. A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science*, 5(5):814–821, 1974.

[17] N. Megiddo. On Finding Primal- and Dual-Optimal Bases. *ORSA Journal on Computing*, 3(1):63–65, 1991.

[18] R. Mohamed. Efficient Dual Simplex Optimizers for Generalized Network Models. unpublished dissertation, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1995.

[19] J. Mulvey and S. Zenios. Solving Large Scale Generalized Networks. *Journal of Information and Optimization Sciences*, 6(1):95–112, 1985.

[20] K. Murty. *Linear and Combinatorial Programming.* Robert E. Krieger Publishing Company, Malabar, Florida, 1985.

[21] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1988.

[22] W. Nulty and M. Trick. GNO/PC Generalized Network Optimization System. *Operations Research Letters*, 7(2):101–102, 1988.

[23] R. Parker and R. Rardin. *Discrete Optimization.* Academic Press, New York, 1988.

[24] Using the CPLEX Callable Library Version 3.0. Incline Village, Nevada 89451-9436, 1994.

Appendix D

Distribution List

Donald Wagner ONR 311 Ballston Centre Tower One 800 North Quincy Street Arlington, VA 22217–5660	3 copies
Administrative Grants Officer Office of Naval Research Regional Office 4520 Executive Drive Suite 300 San Diego, CA 92121–3019	1 copy
Director, Naval Research Laboratory Attn: Code 2627 4555 Overlook Drive Washington, DC 20375–5326	1 copy
Defense Technical Information Center 8725 John J. Kingman Road SDTS 0944 Ft. Belvoir, VA 22060–6218	2 copies
Carol Voltmer Office of Research Administration SMU Dallas, TX 75275	1 copy